(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 December 2001 (13.12.2001)

PCT

(10) International Publication Number
WO 01/95089 A2

(51) International Patent Classification[7]: G06F 5/06

(21) International Application Number: PCT/US01/18667

(22) International Filing Date: 8 June 2001 (08.06.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/210,642 9 June 2000 (09.06.2000) US

(71) Applicant *(for all designated States except US)*: **THE TRUSTEES OF COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK** [US/US]; 116th Street and Broadway, New York, NY 10027 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **CHELCEA, Tiberiu** [RO/US]; 424 West 119th Street, Apt. 67, New York, NY 10027 (US). **NOWICK, Steven, M.** [US/US]; 145 Howard Terrace, Leonia, NJ 07605 (US).

(74) Agents: **TANG, Henry** et al.; Baker Botts LLP, 30 Rockefeller Plaza, New York, NY 10112-0228 (US).

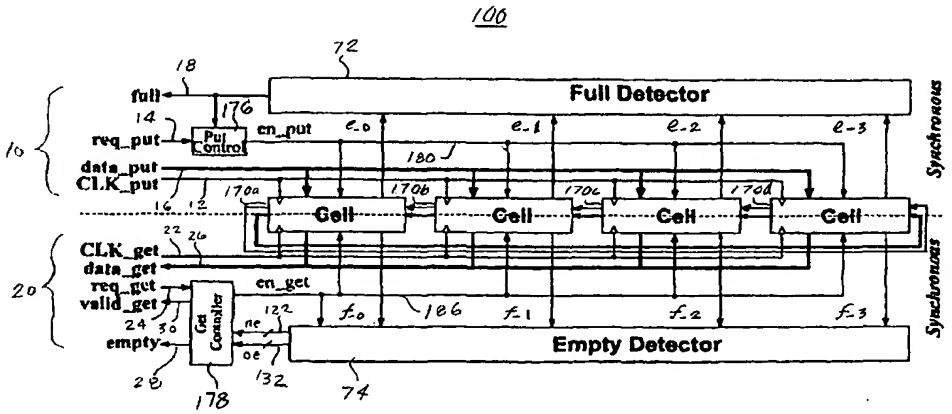
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: LOW LATENCY FIFO CIRCUITS FOR MIXED ASYNCHRONOUS AND SYNCHRONOUS SYSTEMS



(57) Abstract: A FIFO design interfaces a sender subsystem and a receiver subsystem operating on different time domains. The sender subsystem and the receiver subsystem may be synchronous or asynchronous. The FIFO circuit includes a put interface configured to operate in accordance with the sender time domain and get interface configured to operate in accordance with the receiver time domain. The FIFO circuit includes an array of cells having a register and state controller indicative of the state of the cell. Each cell also has a put component part configured to operate according to the sender time domain including a put token passing circuit and put controller circuit. Each cell has get component part configured to operate according to the receiver time domain including a get token passing circuit and a get controller circuit. A mixed-clock relay station design interfaces a sender subsystem and a receiver subsystem working at different time domains, and where the latency between sender and receiver is large.

WO 01/95089 A2

LOW LATENCY FIFO CIRCUITS FOR MIXED ASYNCHRONOUS AND SYNCHRONOUS SYSTEMS

SPECIFICATION

CROSS-REFERENCE TO RELATED APPLICATION

This application claims priority to U.S. Provisional Patent Application entitled "Low-Latency FIFO For Mixed-Clock Systems," Serial No. 60/210,642, which was filed on June 9, 2000, which is incorporated by reference in their entirety herein.

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

This invention relates to FIFO circuits, and more particularly to low latency FIFO designs that interface subsystems working at different speeds and that may be synchronous or asynchronous, and between subsystems with very long interconnection delays.

BACKGROUND OF RELATED ART

A trend in VLSI is increasingly towards a "system-on-a-chip" involving many clock domains. A challenging problem is to robustly interface these domains. There have been few adequate solutions, especially ones providing reliable low-latency communication.

There are two fundamental challenges in designing systems-on-a-chip. A first challenge concerns systems operating under different timing assumptions. These timing assumptions include different clock speeds, as well as both synchronous and asynchronous environments. A second challenge concerns designs having long delays in communication between systems.

A number of FIFO circuits and components have been developed to handle timing discrepancies between subsystems. Some designs are limited to handling single-clock systems. These approaches have been proposed to handle clock

skew, drift and jitter (R. Kol et al., "Adaptive Synchronization for Multi-Synchronous System," *IEEE International Conference on Computer Design (ICCD'98)*, pp.188-189, October 1998; and M. Greenstreet, "Implementing a STARI Chip," *Proceedings IEEE International Conference on Computer Design (ICCD)*, pp. 38-43, 1995). To handle long interconnect delays, "latency - insensitive protocols" have been proposed (L. Carloni et al., "A Methodology for Correct-by-Construction Latency Insensitive Design," *ICCAD,* 1999); however their solution was limited to a single clock domain.

Several designs have also been proposed to handle mixed-timing domains. One category of design approaches attempts to synchronize data items and/or control signals with the receiver, without interfering with its clock. In particular, Seizovic robustly interfaces asynchronous with synchronous environments through a "synchronization FIFO". (J. Seizovic, "Pipeline Synchronization," *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems,* pp. 87-96, November 1994). However, the latency of this design is proportional with the number of FIFO stages, whose implementation include expensive synchronizers. Furthermore, his design requires the sender to produce data items at a constant rate.

Other designs achieve robust interfacing of mixed-clock systems by temporarily modifying the receiver's clock. Synchronization failures are avoided by pausing or stretching the receiver's local clock. Each communicating synchronous system is wrapped with asynchronous logic, which is responsible for communicating with the other systems and for adjusting the clocks. This approach changes the local systems' clocks, and may introduce latency penalties in restarting them.

Jerry Jex et al. U.S. Patent No. 5,598,113 describes a mixed-clock FIFO circuit. However, the FIFO circuit described in '113 has a significantly greater area overhead in implementing the synchronization. For example, this design has two synchronizers for every cell.

Accordingly, there exists a need in the art for a FIFO circuit having low latency and high throughput and capable of operation in mixed synchronous / asynchronous environments.

SUMMARY OF THE INVENTION

An object of the present invention is to provide a FIFO circuit having low latency and high throughput.

Another object of the invention is to provide a FIFO circuit useful in mixed synchronous / asynchronous environments.

A further object of the present invention is to provide a FIFO circuit which may be used as a relay station in connection with long delays and mixed synchronous / asynchronous environments.

A still further object of the present invention is to provide FIFO circuit components which are configured for use in particular protocol of operation, i.e., synchronous or asynchronous, and which are capable of being used in connection with other components regardless of the protocol of operation of the other components.

These and other objects of the invention which will become apparent with respect to the disclosure herein, are accomplished by a FIFO circuit which interfaces the transmission of data items between a sender subsystem operating under a first protocol of operation and a receiver subsystem operating under a second protocol of operation.

A put interface is configured to operate according to the first protocol of operation and comprises a put data bus to transmit a data item from the sender subsystem and a put data request input to receive a put request from the sender subsystem to enqueue the data item from the put data bus. A get interface is configured to operate according to the second time domain and comprises a get data bus to transmit the data item to the receiver subsystem and a get data request input to receive a get request from the receiver subsystem to dequeue the data item to the get data bus.

An array of cells is provided. Each cell has a register configured to receive the data item from the put data bus and to transmit the data item to the get data bus, a state controller providing an indication of the state of the cell, a put component configured to operate according to the first protocol of operation and a get component configured to operate according to the second protocol of operation.

The put component receives the put token from a first adjacent cell, latches the data item received from the put data bus to the register based on the put request, the put token, and the state of the cell, and passes the put token to a second adjacent cell. The get component receives the get token from the first adjacent cell, dequeues the data item from the register to the get data bus based on the get request, the get token, and the state of the cell, and passes the get token to the second adjacent cell.

In accordance with the invention, the objects as described above have been met, and the need in the art for a FIFO circuit having low latency and high throughput and capable of operation in mixed synchronous / asynchronous environments has been substantially satisfied. Further features of the invention, its nature and various advantages will be more apparent from the accompanying drawings and the following detailed description of illustrative embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic view of a synchronous put interface in accordance with the invention.

FIG. 2 is a schematic view of a synchronous get interface in accordance with the invention.

FIG. 3 is a schematic view of an asynchronous put interface in accordance with the invention.

FIG. 4 is a schematic view of an asynchronous get interface in accordance with the invention.

FIG. 5 is a schematic block diagram of an exemplary FIFO circuit in accordance with a first embodiment of the invention.

FIG. 6 is a more detail schematic block diagram of the FIFO circuit illustrated in FIG. 5.

FIG. 7 is a schematic block diagram of an exemplary FIFO circuit in accordance with a second embodiment of the invention.

FIG. 8 is a more detail schematic block diagram of the FIFO circuit illustrated in FIG. 7.

FIG. 9 is a schematic block diagram of an exemplary FIFO circuit in accordance with a third embodiment of the invention.

FIG. 10 is a more detail schematic block diagram of the FIFO circuit illustrated in FIG. 9.

FIG. 11 is a time plot of exemplary signals applied in connection with a synchronous put protocol in accordance with the invention.

FIG. 12 is a time plot of exemplary signals applied in connection with a synchronous get protocol in accordance with the invention.

FIG. 13 is a time plot of exemplary signals applied in connection with an asynchronous put protocol in accordance with the invention.

FIG. 14 is a time plot of exemplary signals applied in connection with an asynchronous get protocol in accordance with the invention.

FIG. 15 is an enlarged schematic block diagram, illustrating a portion of the FIFO circuit of FIGS. 5 and 6, in accordance with the invention.

FIG. 16 is a more detailed schematic block diagram of the portion of the FIFO circuit illustrated in FIG. 15, in accordance with the invention.

FIG. 17 is a schematic block diagram of a full detector of the FIFO circuit illustrated in FIGS 5 and 6, in accordance with the invention.

FIG. 18 is a schematic block diagram of a first empty detector of the FIFO circuit illustrated in FIGS. 5 and 6, in accordance with the invention.

FIG. 19 is a schematic block diagram of a second empty detector of the FIFO circuit illustrated in FIGS. 5 and 6, in accordance with the invention.

FIG. 20 is a schematic block diagram of another portion of the FIFO circuit illustrated in FIGS. 5 and 6, in accordance with the invention.

FIG. 21 is a schematic block diagram of a further portion of the FIFO circuit illustrated in FIGS. 5 and 6, in accordance with the invention..

FIG. 22 is an enlarged schematic block diagram, illustrating a portion of the FIFO circuit of FIGS. 7 and 8, in accordance with the invention.

FIG. 23 is a more detailed schematic block diagram of the portion of the FIFO circuit illustrated in FIG. 22, in accordance with the invention.

FIG. 24 is a burst-mode specification of a portion of the FIFO illustrated in FIG. 23, in accordance with the invention.

FIG. 25(a) is a more detailed schematic block diagram of a portion of the FIFO circuit illustrated in FIG. 23 in accordance with the invention.

FIGS. 25(b) is a more detailed schematic block diagram of another embodiment of a portion of the FIFO circuit illustrated in FIG. 23 in accordance with the invention.

FIG. 26 is a more detailed schematic block diagram of a further portion of the FIFO circuit illustrated in FIG. 23 in accordance with the invention.

FIG. 27 is a more detailed schematic block diagram of a still further portion of the FIFO circuit illustrated in FIG. 23 in accordance with the invention.

FIG. 28 is a Petri-net specification of a portion of the FIFO circuit illustrated in FIG. 23 in accordance with the invention.

FIG. 29 is an enlarged schematic block diagram, illustrating a portion of the FIFO circuit of FIGS. 9 and 10, in accordance with the invention.

FIG. 30 is a more detailed schematic block diagram of the portion of the FIFO circuit illustrated in FIG. 29, in accordance with the invention.

FIG. 31(a) is a more detailed schematic block diagram of a portion of the FIFO circuit illustrated in FIG. 30 in accordance with the invention.

FIGS. 31(b) is a more detailed schematic block diagram of another embodiment of a portion of the FIFO circuit illustrated in FIG. 30 in accordance with the invention.

FIG. 32 is a burst-mode specification of a portion of the FIFO illustrated in FIG. 30, in accordance with the invention.

FIG. 33 is a more detailed schematic block diagram of a further portion of the FIFO circuit illustrated in FIG. 30 in accordance with the invention.

FIG. 34 is a more detailed schematic block diagram of a still further portion of the FIFO circuit illustrated in FIG. 30 in accordance with the invention.

FIG. 35 is a Petri-net specification of a portion of the FIFO circuit illustrated in FIG. 30 in accordance with the invention.

FIG. 36 is a schematic block diagram of a prior art system.

FIG. 37 is a schematic block diagram of a prior art system incorporating relay stations.

FIG. 38 is a schematic block diagram of a prior art relay station.

FIG. 39 is a schematic block diagram illustrating a FIFO circuit relay station in accordance with the invention.

FIG. 40 is a more detailed schematic block diagram illustrating a FIFO circuit relay station illustrated in FIG. 39 in accordance with the invention.

FIG. 41 is detailed schematic view of a portion of the FIFO circuit relay station illustrated in FIG. 40 in accordance with the invention.

FIG. 42 is detailed schematic view of another portion of the FIFO circuit relay station illustrated in FIG. 40 in accordance with the invention.

FIG. 43 is a schematic block diagram illustrating another FIFO circuit relay station system in accordance with the invention.

FIG. 44 is a more detailed schematic block diagram illustrating the FIFO circuit relay station system of FIG. 43 in accordance with the invention.

FIG. 45 is a more detailed schematic block diagram illustrating a FIFO circuit relay station illustrated in FIG. 44 in accordance with the invention.

FIG. 46 is detailed schematic view of a portion of the FIFO circuit relay station illustrated in FIG. 45 in accordance with the invention.

FIG. 47 is a schematic block diagram illustrating yet another FIFO circuit relay station system in accordance with the invention.

FIG. 48 is a more detailed schematic block diagram illustrating the FIFO circuit relay station system of FIG. 47 in accordance with the invention.

FIG. 49 is a more detailed schematic block diagram illustrating a FIFO circuit relay station illustrated in FIG. 48 in accordance with the invention.

FIG. 50 is a detailed schematic view of a portion of the FIFO circuit relay station illustrated in FIG. 49 in accordance with the invention.

DETAILED DESCRIPTION OF THE EXEMPLARY EMBODIMENTS

The FIFO circuits in accordance with the invention mediate between two subsystems: a sender subsystem which produces data items and a receiver subsystem which consumes data items. The FIFO circuits are implemented as a

circular buffer of identical cells, in which each cell communicates with the two subsystems on common data buses. The input and output behavior of a cell is dictated by the flow of two tokens around the ring: one for enqueuing data and one for dequeuing data. Data items are not moved around the ring once they are enqueued, thus providing the opportunity for low-latency: once a data item is enqueued, it is shortly thereafter available to be dequeued.

Each FIFO circuit is partitioned into modular components which may be used with other modular components in a number of different systems, as will be described below. A set of interfaces, or portions of the FIFO circuit, can be combined together to obtain complete FIFO circuits which meet the desired timing assumptions on both the sender's and receiver's end, as will be described herein. As used herein, "time domain" shall refer to whether the subsystem is synchronous or asynchronous.

In accordance with the invention, each FIFO circuit has two interfaces, or portions of the FIFO circuit which are designed to cooperate with the environment. First, the put interface is the portion of the FIFO circuit which communicates with the sender subsystem. A synchronous put interface is illustrated in FIG. 1, and an asynchronous put interface is illustrated in FIG. 3. Second, the get interface is the portion of the FIFO circuit which communicates with the receiver subsystem. FIG. 2 illustrates a synchronous get interface, and FIG. 4 illustrates an asynchronous get interface. A FIFO circuit which interfaces with the environment, whether both the sender and receiver subsystems are synchronous, synchronous and asynchronous, or asynchronous and synchronous, respectively, may be assembled by attaching a put interface and a get interface which properly correspond to the conditions presented.

FIGS. 1-2 illustrate two synchronous interfaces in accordance with the invention. The synchronous put interface 10, illustrated in FIG. 1, is controlled by sender clock signal 12 (*CLK_put*). There are two inputs to the synchronous put interface 10, a put request signal 14 (*req_put*), which communicates requests by the sender to enqueue data and a sender data bus 16 (*data_put*), which is the bus for data items. The global full signal 18 (*full*) is only asserted when the FIFO circuit is full, otherwise it is de-asserted.

FIG. 2 illustrates the synchronous get interface 20, which is controlled by receiver clock signal 22 (*CLK_get*). A single control input from the receiver to dequeue a data item is get request signal 24 (*req_get*). Data is placed on the receiver data bus 26 (*data_get*). A global empty signal 28 (*empty*) is asserted only when the FIFO circuit is empty. The valid data signal 30 (*valid_get*) indicates that the data item is valid.

The asynchronous interfaces, illustrated in FIGS. 3-4, are not synchronized to a clock signal. FIG. 3 illustrates an asynchronous put interface 40 has two inputs which are substantially similar to the inputs of synchronous put interface 10 (FIG. 1). First, a put request signal 44 (*put_req*) is provided, which communicates requests by the sender to enqueue data, and second, a sender data bus 46 (*put_data*) is provided for the output of data items. In contrast to synchronous put interface 10, this interface does not have a global full signal; instead, the asynchronous put interface 40 provides a put acknowledgement signal 45 (*put_ack*) when the put operation is completed.

The asynchronous get interface 50, illustrated in FIG. 4, has a get request signal 54 (*get_req*) and an output data bus 58 (*get_data*). Unlike the synchronous counterpart, this interface does not have a data validity signal or a global empty signal. The asynchronous get interface 50 provides a get acknowledgement signal 55 (*get_ack*), which indicates that the get operation is completed.

The modular interfaces 10, 20, 40, and 50 of FIGS. 1, 2, 3, and 4, respectively, may be attached in accordance with the invention to create FIFO circuits which may transmit data between different environments. For example, a FIFO circuit which interfaces between a synchronous sender subsystem and a synchronous receiver subsystem may be formed by using the synchronous put interface 10 and the synchronous get interface 20, in order to form a FIFO circuit referred to herein as a "synch-synch" FIFO circuit, as will be described with respect to exemplary embodiment FIFO circuit 100. Similarly, a FIFO circuit that interfaces between an asynchronous sender subsystem and a synchronous receiver subsystem would incorporate asynchronous put interface 40 along with synchronous get interface 20, which may be referred to as an "asynch-synch" FIFO, and will be described in greater

detail with respect to exemplary embodiment FIFO circuit 200. Likewise, to interface between a synchronous sender subsystem and an asynchronous receiver subsystem, which may be referred to as a "synch-asynch" FIFO, may utilize synchronous put interface 10 and asynchronous get interface 50, as described herein with respect to exemplary embodiment FIFO circuit 300.

FIFO circuits 100, 200, and 300 of FIGS. 6, 8, and 10, respectively, are substantially similar, with the differences noted herein. For example, each FIFO circuit 100, 200, and 300 has a circular array of identical cells which communicate with the put and get interfaces on common data buses. The control logic for each operation is distributed among the cells, and allows concurrency between the two interfaces. Data is immobile in the array of cells. Consequently, once a data item is enqueued, it is not moved, and is simply dequeued in place.

At any time, there are two tokens in the FIFO circuits 100, 200, and 300, i.e., a put token and a get token. The input and output behavior of the FIFO circuits 100, 200 and 300 is controlled by these tokens. The put token is used to allow the enqueuing of data items, and the get token is used to allow the dequeuing of data items. A cell having the put token may be considered the "tail" of the queue, and the cell having the get token may be considered the "head" of the queue. In normal operation, the put token is typically ahead of the get token. Once a cell has used a token for a data operation, the token is passed to the next cell after the respective operation is completed. The token movement is controlled both by interface requests as well as by the state of the FIFO circuit, i.e., empty or full, as will be described in greater detail below.

There are several advantages that are common to the architectures of FIFO circuits 100, 200, and 300. Since data is not passed between the cells from input to output, the FIFO circuits have a potential for low latency. Consequently, as soon as a data item is enqueued, it is also available for dequeuing. Secondly, the FIFO circuits offer the potential for low power: data items are immobile while in the FIFO circuit. Finally, these architectures are highly scalable; the capacity of the FIFO circuit and the width of the data item can be changed with very few design modifications.

The FIFO circuit 100 in accordance with a first exemplary embodiment is illustrated in FIGS. 5 and 6. The FIFO circuit 100 may be used when the sender subsystem is synchronous and the receiver subsystem is also synchronous. Consequently, the modular synchronous put interface 10 is used in connection with the modular synchronous get interface 20, as illustrated in FIG. 5. The sender subsystem operates on the sender clock signal 12 (*CLK_put*), and the receiver subsystem operates on the receiver clock signal 22 (*CLK_get*).

As illustrated in FIG. 6, FIFO circuit 100 is constructed with a circular array of identical cells 170a, 170b, 170c, and 170d, and communicates with the two external interfaces, i.e., the synchronous put interface 10 on sender data bus 16 (*data_put*) and the synchronous get interface 20 on receiver data bus 26 (*data_get*).

The synchronous interfaces 10 and 20 have two additional types of components: (1) detectors, which determine the current state of the FIFO circuit 100, i.e., empty or full, and (2) external controllers, which conditionally pass requests for data operations to the cell array. As is known in the art, a data operation on a synchronous interface is completed within a clock cycle; therefore, the environment does not need an explicit acknowledgement signal. However, if the FIFO circuit 100 becomes full (or empty), the environment may need to be stopped from communicating on the put (or get) interface. Detectors and controllers operate in the FIFO circuit 100 to detect the exception cases, and stall the respective interface if it is not safe to perform the data operation. As illustrated in FIG. 6, the full detector 72 and empty detector 74 observe the state of all cells 170a, 170b, 170c, and 170d and compute the global state of the FIFO circuit 100, i.e., full or empty. The output of the full detector 72 may passed to the put interface 10, while the output of the empty detector 72 may be passed to the get interface 20. The put controller 176 and get controller 178 filter data operation requests to the FIFO circuit 100. Thus, the put controller 176 usually passes put requests from the sender subsystem, but disables them when the FIFO circuit is full. Similarly, the get controller 178 normally forwards get requests from the receiver subsystem, but blocks them when the FIFO circuit 100 is empty. The detectors 72 and 74, the external controllers 176 and 178, and the definitions of "empty" and "full" will be described in greater detail below.

The FIFO circuit 200 in accordance with a second exemplary embodiment is illustrated in FIGS. 7 and 8. The FIFO circuit 200 may be used when the sender subsystem is asynchronous and the receiver subsystem is synchronous. Consequently, the modular asynchronous put interface 40 is used in connection with the modular synchronous get interface 20, as illustrated in FIG. 7. The sender subsystem operates asynchronously, while the receiver subsystem operates on the receiver clock signal 22 (*CLK_get*).

As illustrated in FIG. 8, FIFO circuit 200 is constructed with a circular array of identical cells 270a, 270b, 270c, and 270d, and communicates with the two external interfaces, the asynchronous put interface 40 and the synchronous get interface 20 on common data buses: a sender data bus 46 (*data_put*) and receiver data bus 16 (*get_data*).

The synchronous get interface 20, as described above with respect to FIG. 6, includes an empty detector 74 and a get controller 278. The get controller 278 typically passes get requests from the receiver subsystem, but disables such requests when the FIFO circuit 200 is empty. In contrast to synchronous interfaces, asynchronous interfaces, such as asynchronous put interface 40, do not include full or empty detectors or external controllers. Since an asynchronous interface does not operate on a clock signal, it does not need to raise an exception, such as "full" or "empty," to temporarily stall data operation. Therefore, when a FIFO circuit having an asynchronous interface becomes full (or empty), the put (or get) acknowledgement can be withheld indefinitely until it is safe to perform the data operation. In the case of FIFO circuit 200, the put acknowledgement signal 45 (*put_ack*) is withheld when the FIFO circuit 200 is full, and transmitted when it is safe to perform the put operation.

FIGS. 9 and 10 illustrate the FIFO circuit 300 in accordance with a third exemplary embodiment. The FIFO circuit 300 may be used when the sender subsystem is synchronous and the receiver subsystem is asynchronous. The synchronous put interface 10 is used in connection with the asynchronous get interface 50, as illustrated in FIG. 9. The sender subsystem operates on the sender clock signal 12 (*CLK_put*), while the receiver subsystem operates asynchronously.

As illustrated in FIG. 10, FIFO circuit 300 contains a circular array of identical cells 370a, 370b, 370c, and 370d, and communicates with the two external interfaces, the synchronous put interface 10 and the asynchronous get interface 50 on common data buses a sender data bus 16 (*data_put*) and receiver data bus 56 (*get_data*).

The synchronous put interface 10, as described above with respect to FIG. 6, includes a full detector 72 and a put controller 376. FIFO circuit 300 does not have an empty detector; instead, the get acknowledgement signal 55 (*get_ack*) is withheld when the FIFO circuit 300 is empty, and transmitted when it is safe to perform the get operation.

The synchronous put protocol is illustrated in FIGS. 11, and discussed in connection with FIG. 1. The synchronous put interface 10 starts a put operation when it receives a put request signal 14 (*put_req*) and a data item on the put data bus 16 (*data_put*), immediately after the positive edge of sender clock signal 12 (*CLK_put*). The data item is enqueued immediately after the positive edge of the next clock cycle (not shown). If the FIFO circuit becomes full, then the global full signal 18 (*full*) is asserted before the next clock cycle, as illustrated in FIG. 11, and the synchronous put interface 10 is prevented from any further operation.

FIG. 12 illustrates the synchronous get protocol, which is discussed in connection with FIG. 2, above. A synchronous get operation is enabled by a get request signal 24 (*req_get*), which is asserted immediately after the positive edge of receiver clock signal 22 (*CLK_get*). By the end of the clock cycle, a data item is placed on get data bus 28 (*data_get*) together with its validity signal 30 (*valid_get*). If the FIFO circuit becomes empty during that clock cycle, then the global empty signal 28 (*empty*) is also asserted immediately after the positive edge of the next clock cycle, and the synchronous get interface 20 is stalled until the FIFO circuit becomes non-empty. Following a get request 24 (*req_get*), then validity signal 30 (*valid_get*) and global empty signal 28 (*empty*) can indicate three outcomes: (a) a data item may be dequeued, and more data items available (i.e., *valid_get* = 1, *empty* = 0); (b) a data item may be dequeued, and FIFO circuit has become empty (i.e., *valid_get* = 1, *empty*

= 1); or (c) the FIFO circuit is empty, and no data item is dequeued (*valid_get* = 0, *empty* = 1).

The asynchronous put and get protocols are illustrated in FIGS. 13 and 14. Since the asynchronous interfaces do not have a clock, they use a 4-phase bundle-data style of communication, as is known in the art (Further details on bundle-data communication are described in S. Furber, "Asynchronous Design," *Proceedings of Submicron Electronics*, pp. 461-492, 1997; I. Sutherland, "Micropipelines," *Communications of the ACM*, 32(6), pp. 720-738, June 1989, and H. van Gageldonk et al., "An Asynchronous Low-Power 805C51 Microcontroller," *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems,* pp. 96-107, 1998, which are incorporated by reference in their entirety herein.) Data items must have stable values on the data buses before a data operation is requested.

As illustrated in FIG. 13, the sender starts a put operation by placing a data item on the sender data bus 46 (*put_data*) and issuing a request to the FIFO circuit to enqueue it by asserting put request signal 44 (*put_req*). The completion of the enqueuing operation is indicated by asserting put acknowledgement signal 45 (*put_ack*). The two control wires are subsequently reset to the idle state, first put request 44 (*put_req*) and then put acknowledgement 45 (*put_ack*).

Similarly, an asynchronous get operation is illustrated in FIG. 14. The get operation is performed by asserting the get request signal 54 (*get_req*) and by dequeuing a data item onto the get data bus 56 (*get_data*). Upon completion of this operation, the get acknowledgement signal 55 (*get_ack*) is asserted. Subsequently, the get request signal 54 (*get_req*) is de-asserted and then the get acknowledgement signal 55 (*get_ack*) is also de-asserted.

In order to construct FIFO circuits 100, 200 and 300 which operate correctly with synchronous and asynchronous systems, each cell has a configuration that consists of four distinct, interchangeable component parts that are selected to interface with the sender or receiver environment: (1) a put component that performs the put operation and is configured to operate with the sender environment, i.e., synchronous or asynchronous, (2) a get component that performs the get operation

and is configured to operate with the receiver environment, i.e., synchronous or asynchronous, (3) a data validity (DV) controller which provides an indication of whether the cell has a data item, and is configured to operate with both the sender and receiver environments, and (4) a register which is configured to operate with both the sender and receiver environments. Consequently, the put components in cells 170a-d and 370a-d will be substantially identical because they are configured to operate with a synchronous sender environment. The get components in cells 170a-d and 270a-d will be substantially identical because they are configured to operate with a synchronous receiver environment. The put components of cells 270a-d are configured to operate with an asynchronous sender environment, and the get components of cells 370a-d are configured to operate with an asynchronous receiver environment.

The purpose of the data validity controller is to indicate when the cell is full and when it is empty, and when it has valid data. The register in each cell is split into two parts, one belonging to the put component (the write port), and one belonging to the get component (read port). The put component, the get component, the data validity controller, and the register are attached together to obtain a complete cell.

The FIFO circuit 100, which may be used in connection with a synchronous sender and a synchronous receiver, was described above along with an array of identical cells, 170a, 170b, 170c, and 170d in connection with FIGS. 5 and 6. An exemplary individual cell, such as cell 170a, of FIFO circuit 100 is illustrated in FIGS. 15 and 16. (The following description for cell 170a also applies to cells 170b, 170c, and 170d.) Each cell 170a has four interfaces: (1) a synchronous put interface 171, (2) a synchronous get interface 172, (3) an interface 173 with a right cell in the array of cells, and (4) an interface 174 with a left cell in the array of cells. On the synchronous put interface 171, the cell 170a receives data on the common put data bus 16 (*data_put*). It is enabled to perform a put operation by the put enable signal 80 (*en_put*), which is the output of the put controller 76 (See FIG. 6). The put request signal 14 (*req_put*) indicates data validity (which is always asserted in this embodiment). The passing of put request signal 14 (*req_put*) to cell 170a has been

omitted from FIG. 6, above, to avoid complicating the figure. The cell 170a communicates with the full detector 72, with empty bit 182 (*e_i*), which is asserted when the cell 170a is empty. The state of the exemplary cell 170a is indicated by the data validity (DV) controller, such as asynchronous SR latch 180. On the synchronous get interface 172 (FIG. 15), the cell 170a outputs data on the common get data bus 26 (*data_get*) together with its validity signal 184 (*valid_i*), which is always asserted in this embodiment. As illustrated in FIG. 6, the validity signal 184 (*valid_i*) is not used in FIFO circuit 100, but will be used in the several embodiment described below. The synchronous get interface 172 is enabled by get enable signal 186 (*en_get*), which is the output of get controller 78 (See FIG. 6). The cell 170a communicates with the empty detector 74 with the full bit 188a (*f_i*), which is asserted when the cell 170a is full. On the interface 173 with the right cell, each cell 170a-d receives tokens on put token input 190 (*ptok_in*) and receives the get token input 192 (*gtok_in*) from the right cell. On the interface 174 with the left cell, each cell 170 passes the tokens on put token output 194 (*ptok_out*) and the get token output 196 (*gtok_out*) to the left cell.

Cell 170a in accordance with the first embodiment is also shown in FIG. 16. The behavior of cell 170a may be illustrated by tracing a put operation and then a get operation through the cell 170a. Initially, the cell 170a starts in an empty state (i.e., $e\_i = 1$ and $f\_i = 0$) and without any tokens. The cell 170a waits to receive the put token on put token input 190 (*ptokn_in* = 1) from the right cell on the positive edge of sender clock signal 12 (*CLK_put*), and waits for the sender to place a valid data item on the put data bus 16 (*data_put*). A valid data item is indicated to all cells by the put enable signal 180 (*en_put* = 1), which is the output of the put controller 176 (See FIG. 6).

When there is valid data and the cell has obtained the put token (i.e., AND 181), the cell 170a performs three actions: (1) it enables the register 191 (*REG*) to latch the data item and also the put request signal 14 (*req_put*); (2) it indicates that the cell 170a has a valid data item (asynchronously sets $f\_i = 1$); and (3) it enables the upper left edge-triggered D-type flip-flop ETDFF 193 (*en_put* = 1) to pass the put token to the left cell on the put token output 194 *ptok_out*. On the positive edge of the

next clock cycle of the sender clock signal 12 (*CLK_put*), the data item and validity bit are finally latched and the put token is passed to the left cell.

Dequeuing data by cell 170a proceeds in a substantially identical manner, which the differences noted herein. The cell 170a waits to receive the get token on get token input 192 (*gtok_in* = 1) from the right cell. When this occurs, cell 170 enables the broadcasting of the valid bit 183 (*v_i*), i.e., the latched put request signal 14 (*req_put*), onto the valid bus 184 (*valid_i*). When both the get token is received on get token input 192 (*gtok_in* = 1) and the receiver requests a data item with the get enable signal 186 (*en_get* = 1), as implemented by the AND gate 181, the cell 170a asynchronously enables the data item to be placed on the common get data bus 26 (*data_get*) and indicates that the cell 170 is empty (asynchronously sets *e_i* = 1). The arrival of the asserted get enable 186 (*en_get*) enables the lower left edge-triggered D-type flip-flop ETDFF 195 to pass the get token on the get token output 196 (*gtok_out*). At the beginning of the next clock cycle, the get token is then passed to the left cell.

Each of the FIFO circuits 100, 200, and 300 may have at least one synchronous interface. Therefore, the FIFO circuit operations must be synchronized. A mixed-clock FIFO circuit has highly concurrent operation: at any time, the FIFO circuit's state, i.e., full or empty, may be modified by either the put interface 10 and/or the get interface 20, each of which may be operating under a different clock or asynchronously. At the same time, each interface "reads" the state of the FIFO circuit under its own clock. The global full signal 18 (*full*) (FIG. 6 and 10) is read by the put interface, and the global empty signal 28 (*empty*) (FIG. 6 and 8) is read by the get interface. Therefore, to avoid inconsistent reads, synchronizers have been added to each of the two global control signals, full signal 18 (*full*) and empty signal 28 (*empty*). Each synchronizer conditions the corresponding global signal to the appropriate clock. In the exemplary embodiment, a pair of synchronizing latches is used; however, for increased robustness, it is noted that more than two latches may be used. As illustrated in FIGS. 17, 18 and 19, below, synchronizers are added to the output of the full detector 72 and the empty detector 74, and are controlled by the

sender clock signal 12 (*CLK_put*) and the receiver clock signal 22 (*CLK_get*), respectively.

The synchronizers described herein add additional clock cycles of delay to the reading of the current state of the FIFO circuit. Consequently, simple full and empty detectors which merely indicate the immediate state of the FIFO circuit may result in failure, i.e., overflow or underflow. For example, when the FIFO circuit using a pair of synchronizing latches becomes full, the sender interface is stalled two clock cycles later. In the next clock cycle, the sender might deposit a new data item, effectively overwriting a unread data item. Conversely, when the FIFO circuit becomes empty, the receiver interface is stalled two clock cycles later, so in the next clock cycle the receiver might read an empty cell.

A solution in accordance with the invention is to modify the definition and implementation of the global full signal 18 (*full*) and the global empty signal 28 (*empty*), to anticipate an "imminent" full or empty state, to stop the interfaces in time, but not prematurely or too late. According to the definition, the FIFO circuit is considered "full" when fewer than a predetermined number of cells are empty. (The definition of "empty" is considered in greater detail below.) In accordance with the first exemplary embodiment, the FIFO circuit is considered full when either no cells or one cell is empty. Thus, when there are fewer than two empty cells, the FIFO circuit is declared full, and the sender subsystem can safely deposit a final data item and issue a new unanswered request, before stalling two clock cycles later. The protocols described above with respect to FIGS. 11-12 are unchanged. The only effect may be that sometimes the two systems may see an n-place FIFO circuit as a n-1 place one.

The full detector 72, illustrated in FIGS. 17, implements the definition of "full" described above. The FIFO circuit 100 is declared full when fewer than a predetermined number of consecutive cells are empty. As illustrated in FIG. 17, the empty bits 182 (*e_i*) for consecutive cells are evaluated, i.e., *e_0* and *e_1*; *e_1* and *e_2*; *e_2* and *e_3*; and *e_3* and *e_0*; and the FIFO circuit 100 is declared full if none of these consecutive pairs of cells is found empty. A pair of latches 171 and 173 have

been added to synchronize the global full signal 18 (*full*) with the sender clock signal 12 (*CLK_put*).

A similar definition of "empty" applies when fewer than a predetermined number of cells in the FIFO circuit 100 are full. In the exemplary embodiment, when there are fewer than two data items, the FIFO circuit may be declared empty. Under these circumstances, the receiver subsystem may then remove the last data item and issue a new unanswered request, before stalling two clock cycles later. However, the early detection of empty, as described above, may cause the FIFO circuit 100 to deadlock. A disadvantage of the "nearly empty" (*ne*) definition (zero or one data item(s) in the FIFO circuit), is that the FIFO circuit 100 may be declared empty but nevertheless contains one data item, but the requesting receiver is still stalled.

An alternative definition of empty, as is well known, is "true empty" (*oe*), which is an indication of whether there are *any* data items in the circuit. Since the true empty signal (*oe*) is typically delayed through the synchronization, a disadvantage of the true-empty signal is that it may result in underflow.

A solution in accordance with the invention is to use a bi-modal empty detector 74 (the components of which are illustrated in FIGS. 18-19 as described in greater detail below). The bi-modal detector determines both the "nearly empty" (*ne*) state as well as the "true empty" (*oe*) state. The two empty signals are then synchronized with the receiver and combined into a global empty signal 28 (*empty*). However, the nearly empty definition will normally take precedence over the true empty (*oe*).

The bi-modal empty detector declares the global empty signal 28 (*empty*) based, in part, on the occurrence of recent get requests from the receiver. If there have not been recent get requests, for at least one clock cycle, then the true empty signal (*oe*) dominates. This becomes important when there is one data item in the FIFO circuit 100. The nearly empty signal (*ne*) indicates that the FIFO circuit is empty, and the true empty signal (*oe*) indicates that the FIFO circuit is not empty. In this condition, the get interface 20 needs to receive the data item, so the true empty signal (*oe*) is used to indicate the FIFO state, i.e., "not empty," and is de-asserted.

However, when the get interface has just removed a data item, the nearly empty signal (*ne*) must be used to indicate the state, i.e., "empty," in order to prevent the FIFO underflow, which the synchronization delays for the true empty signal (*oe*) might cause.

According to another scenario, the FIFO circuit may become empty for at least one cycle, i.e., the global empty signal 28 (*empty*) is asserted. During the next clock cycle the true empty signal (*oe*) dominates.

In accordance with the first exemplary embodiment, the bi-modal empty detector 74 is implemented with two detectors, a near-empty detector 120 (FIG. 18) and a true-empty detector 130 (FIG. 19). The two signals are combined to produce the global empty signal 28 (*empty*) as described below with respect to FIG. 19. The near-empty detector 120, illustrated in FIG. 18, is similar to the full detector 72, described above, and evaluates the full bits 188 (*f_i*) of consecutive pairs of cells, i.e., *f_0* and *f_1*, *f_1* and *f_2*, *f_2* and *f_3*, and *f_3* and *f_0*. The FIFO circuit 100 is declared empty on signal 121 (*empty_p*) if no two consecutive pairs of cells are found to have respective data items. A pair of latches 124 and 126 are used to synchronize the near-empty signal 122 (*ne*) to the get clock signal 22 (*CLK_get*).

A true-empty detector 130, illustrated in FIG. 19, determines the "true empty" (*oe*) state of the FIFO circuit. According to this definition, the FIFO circuit 100 is empty if there are zero data items, in which case the true empty signal 132 (*oe*) is asserted. Thus, the true empty detector 130 evaluates each full bit 88 (*f_i*) to determine whether there are any data items present. The true empty signal 132 (*oe*) is delayed through synchronization, which may result in underflow. To prevent this condition, the true empty signal 132 (*oe*) is synchronously set to a neutral "FIFO circuit empty" value after each get operation. This is implemented by OR'ing the output signal 138 after being clocked through the first latch 134 with the get enable signal 186 (*en_get*) before the second latch 136. As a result of setting the "true empty" signal after each get operation, the "true empty" signal (*oe*) does not necessarily indicate that there are no data items in the FIFO circuit. Specifically, if there are more than one data items present and the get enable signal 186 (*en_get*) is asserted, the true-empty detector 130 will output the true empty signal 132 (*oe*).

The potential deadlock problem is solved in accordance with the invention by combination of the near-empty signal 122 and the true-empty signal 132. In most of the cases, the near-empty detector 120 and the true-empty detector 130 produce the same result, i.e., the near-empty signal 122 (*ne*) and the true empty signal 132 (*oe*) are the same. When the FIFO circuit 100 contains a few data items, and the get enable signal 186 (*en_get*) is not asserted during the interval between when latch 134 is clocked and when latch 136 is clocked, both the true empty signal 132 (*oe*) and the near-empty signal 132 (*ne*) indicate the FIFO circuit 100 is not empty, i.e., both signals are de-asserted. Similarly, when the FIFO circuit 100 contains zero data items, both the true empty signal 132 (*oe*) and the near-empty signal 132 (*ne*) indicate the FIFO circuit is empty, i.e., both signals are asserted.

A different situation arises when the FIFO circuit 100 contains *exactly* one data item, i.e., the near-empty signal 122 indicates that the FIFO circuit "empty" ($ne = 1$), and the true empty signal 132 indicates that the FIFO circuit "not empty" ($oe = 0$) in the absence of assertion of the get enable signal 186 (*en_get*) during the aforementioned interval. This condition may arise after the get interface has enabled the removal of the next-to-last data item in the FIFO. The next step will depend upon whether there is another get request: (1) If in the current clock cycle there is another get request, this request is satisfied and the near-empty detector 120 will declare the FIFO empty (i.e., the near empty signal (*ne*) is asserted) and will stall the get interface in the next clock cycle. (2) If there is no get request, then the true empty detector 130 will dominate in the next clock cycle, and declare the FIFO not empty (i.e., the true empty signal (*oe*) is de-asserted), allowing a subsequent get request to be satisfied. Whenever the last data item is dequeued, the near empty signal (*ne*) again immediately dominates and stalls the get interface on time. At this point no further get requests are satisfied, so the near empty signal (*ne*) again is used to indicate the state of the FIFO 100.

The put controller 176 is shown in FIG. 20. The put controller 176 enables and disables the put operation and the movement of the put token in the FIFO circuit 100. As illustrated in FIG. 20, These operations are only enabled when there is a valid data item on *data_put*, i.e., the put request signal 18 (*req_put*) has been

asserted and the FIFO circuit is not full, i.e. the global full signal 18 (*full*) has not been asserted. In the scenario described above, wherein the FIFO circuit becomes empty for at least one cycle, the get enable signal 186 (*en_get*) is de-asserted, regardless of whether get request signals 24 (*req_get*) are made by the receiver. In such case the true empty signal 18 (*oe*) is not re-set in the next clock cycle and is able to dominate. In this case, once the FIFO is empty, if a single data item is enqueued by the put interface, the true empty signal (*oe*) will remain dominant, eventually *oe* will be de-asserted and the global empty signal 28 (*empty*) will in turn be de-asserted, and the get interface 20 will be able to remove the data item, thus avoiding deadlock.

The get controller 178, illustrated in FIG. 21 enables and disables the get operation and the movement of the get token in the FIFO circuit 100. The get enable signal 186 (*en_get*) is only asserted when there is a request from the receiver, i.e., the get request signal 24 (*req_get*) is asserted and at least one of the near-empty detector 120 and true empty detector 130 indicates that the FIFO circuit 100 is not empty.

Each of FIFO circuit 200 (FIGS. 7 and 8) in accordance with the second embodiment and FIFO circuit 300 (FIGS. 9 and 10) in accordance with the third embodiment has one asynchronous interface and one synchronous interface. As described above with respect to FIGS. 7-8, FIFO circuit 200 has an asynchronous put interface and a synchronous get interface, while the FIFO circuit 300 described above with respect to FIGS. 9-10 has a synchronous put interface and an asynchronous get interface. Therefore, each FIFO circuit 200 and 300 utilizes certain synchronous components from FIFO circuit 100 described above. More particularly, the synchronous put interface 10 is used as the put interface in FIFO circuit 300. Similarly, the synchronous get interface 20 in FIFO circuit 100 is also used in FIFO circuit 200 as the get interface.

As described above, each cell has four distinct parts: a put component, a get component, a data validity (DV) controller, and a register. Each of cells 270a-d and 370a-d uses a version of a data validity controller, i.e., data validity controller 280 and data validity controller 380, respectively. In cell 170a (FIG. 16), above, the data validity controller was simple (an SR latch 180). However, for the FIFO circuits 200

and 300 having both synchronous and asynchronous components, the behavior becomes more complex. These designs allow more concurrency between the write operations and the read operations to the same cell. Therefore, the data validity controller has to allow for that increased concurrency, as will be described below.

The interfaces of the FIFO circuit according to the second embodiment, i.e., asynch-synch FIFO circuit 200, were described above with respect to FIG. 7. They are obtained by using the asynchronous put interface 40 (FIG. 3) and the synchronous get interface 20 (FIG. 2). The FIFO circuit protocol on these interfaces was described with respect to FIGS. 12-13, and the architecture of the FIFO circuit 200 was shown in FIGS. 7-8. With specific reference to FIG. 8, FIFO circuit 200 uses a number of components which have been described above, i.e., interface components such as the get controller 278, empty detector 74 and synchronous cell components. The remaining components in cells 270a-d, i.e., the asynchronous put component, and the data validity controller are described herein with respect to FIGS. 22-23.

The synchronous part of cell 270a shown in FIG. 23 is identical to the corresponding components of cell 170a (FIG . 16) in FIFO circuit 100 (FIG. 6). The asynchronous part of cell 270a is decomposed into several blocks. The put component part comprises the ObtainPutToken block 293 (*OPT*) and the C-element 295. The ObtainPutToken block 293 (*OPT*) obtains the respective put token from the right interface on put token input 290 (*we1*). It is implemented as a Burst-Mode machine as illustrated in FIG. 24. (Burst-Mode machines are also described in greater detail in T. Chelcea et al., "Low-Latency Asynchronous FIFO's using Token Rings," *IEEE ASYNCH '00 Symp.*, pp. 210-220, which is incorporated by reference in their entirety herein.) As illustrated in the Burst Mode specification 400 of FIG. 24, the ObtainPutToken block 293 observes the right cell and waits for a put operation. A complete token passing is indicated by the right cell, which first sets the put token input 290 (*we1*) at step 402 and then resets the put token input 290 (*we1*) at step 404. After that operation takes place, the put token is in the current cell, i.e., put token signal 297 (*ptok*) is set (step 404). When the put token signal 297 (*ptok*) is set, another put operation can take place. Once the put operation starts, the put token

signal 297 (*ptok)* is reset and the put token output 294 (*we*) is set at step 406. When the put operation finishes, the put token output 294 (*we*) is reset, the put token is sent to the next cell and the cycle resumes. A first exemplary embodiment of the ObtainPutToken block 293 (*OPT)* is illustrated in FIG. 25(a). A second exemplary embodiment of ObtainPutToken block 293' (*OPT)* is illustrated in FIG. 25(b).

The put operation is controlled by a C-element 295, as illustrated in FIG. 26. As is known in the art, asymmetric C-element 295 has its output at 1 when all its inputs are at 1; the output becomes 0 when all its inputs become zero. In an asymmetric C-element, some of the inputs (marked with '+') participate only in the setting the output of the element to one; their values are irrelevant for the other output transition.

The behavior of cell 270a (FIG. 23) for an asynchronous put operation proceeds as follows: Initially, cell 270a starts in an empty state ($e\_i = 1$ and $f\_i = 0$) and neither the put token or get token are present. After a pair of transitions on put token input 290 (*we*1), the put token is in the cell ($ptok = 1$). When the environment requests a put operation on put request signal 44 ($put\_req = 1$) and the cell is empty ($e\_i = 1$), the put token output 294 (*we*) is set. This event causes several operations in parallel: the state of the cell is changed to full (i.e., the full bit 288a (*f_i*) is asserted) by data validity controller 280 (*DV_as*); the register 291 (*REG*) is enabled to latch data, and cell 270a starts both sending the put token to the left cell and resetting *OPT* ($ptok = 0$). When the put request signal 44 (*put_req*) is de-asserted, the put token output 294 (*we*) is also de-asserted. This event completes the sending of the put token to the left cell. Now cell 270a is prepared to start another put operation after the data in the register 291 (*REG*) is dequeued.

The synchronous get operation in cell 270a starts after a data item is present in the cell. Once the full bit 288a (*f_i)* is set, the empty detector 74 (FIG. 8) computes the state of FIFO circuit 200 as "not empty", and a get request 24 (*req_get*) is passed on the get enable signal 86 (*en_get*). When cell 270a has the get token, then contents of the register 291 (*REG*) are output on the get data bus 26 (*data_get*); on the next positive edge of the receiver clock signal 22 (*CLK_get*), the get token is passed to the next cell. In the same time, the state of the cell is changed to "empty" by the

data validity controller 280 (*DV_as*). Note that when the cell has the get token (*gtok* = 1), the cell's validity bit is broadcast on validity data bus 284 (*valid_i*) regardless of the signal on get request 86 (*en_get*). That ensures that the signal on the validity bus 284 (*valid_i*) is always driven to some value.

As illustrated in FIG. 27, the data validity controller 280 (*DV_as*) indicates when the cell contains data items; it thus controls the put and get operations. It accepts as inputs the put token output 294 (*we*), which signals that a put operation is taking place, and the read enable signal 299 (*re*), which signals that a get operation is taking place. The outputs of the data validity controller 280 (*DV_as*) are the empty bit 282a (*e_i*), indicating cell 270a is empty, (used only internally in this embodiment), and the full bit 288a (*f_i*), indicating cell 270a is full, which is used in the empty detector in this embodiment).

The protocol for data validity controller 280 (*DV_as*) is shown as a Petri-Net 410 in FIG. 28. (A Petri-net is a well-known graphical representation commonly used to describe concurrent behaviors). It consists of transitions, indicated by labeled events, and places, which store tokens which are indicated by black dots. A transition fires when all of its incoming arcs have tokens, which are then deposited on all of its outgoing arcs . (Further details concerning Petri-nets are discussed in Tadao Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE,* 77(4), April 1989; L.Y. Rosenblum and A.V. Yakolev, "Signal Graphs: From Self-Timed to Timed Ones," *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy,* pp. 199-207, July 1985; and Tam-Anh Chu, "On the Models for Designing VLSI Asynchronous Digital Circuits," *Integration, the VLSI Journal,* 4(2):99-113, June 1986, which are incorporated by reference in their entirety herein.) Once a put operation starts, data validity controller 280 (*DV_as*) both resets the empty bit 282a ($e\_i = 0$), and sets the full bit 288a ($f\_i = 1$),thus declaring the cell full enabling a get operation, at 412. After a get operation starts (*re*+) at 414, the cell is declared "not full" ($f_i = 0$) asynchronously at 415, in the middle of the *CLK_get* clock cycle. When the get operation finishes (on the next positive edge of *CLK_get*) at 416, the data validity controller 280 (*DV_as*) sets cell 270a to "empty" ($e_i = 1$) at 417. The put token output 294 (*we*) is reset at 418, and the behavior can resume. This

asymmetric behavior delays the passing of the put token to prevent data corruption by a put operation while a get operation is still taking place.

The FIFO circuit according to the third exemplary embodiment, i.e., synch-asynch FIFO circuit 300 (FIGS. 9 and 10), will be described in greater detail herein. The interfaces of FIFO circuit 300 were described above with respect to FIG. 9. They are obtained by "attaching" a synchronous put interface 10 (See FIG. 1) and an asynchronous get interface 50 (See FIG. 4). The FIFO circuit protocol on these interfaces was described with respect to FIGS. 11 and 14, and the architecture of the FIFO circuit 300 was illustrated in FIGS. 9-10. With reference to FIG. 10, FIFO circuit 300 uses a number of components which have been described above, i.e., interface components such as the put controller 376, full detector 72 and synchronous cell components. The remaining components in cells 370a, e.g., the asynchronous get component, and the data validity controller (*DV_sa)* are described herein with respect to FIGS. 29-30.

The interfaces of exemplary cell 370a are shown in FIG. 29. (Each of cells 370a, 370b, 370c, and 370d are identical.) Cell 370a communicates on four interfaces: (1) a synchronous put interface 371, (2) an asynchronous get interface 372, (3) an interface 373 with the right (i.e., previous) cell to obtain tokens, and (4) an interface 374 with the left (i.e., next) cell to pass the tokens. On the synchronous put interface, cell 370a receives data on the common put data bus 16 (*data_put*). It is enabled to perform a put operation by the put enable signal 80 (*en_put*), which is the output of the put controller 76 (See FIG. 10). The put request signal 14 (*req_put*) indicates data validity (which is always asserted in this embodiment). The cell 370a communicates with the full detector 72 (FIG. 10), with empty bit 82 (*e_i),* which is asserted high when the cell 370a is empty. The put operation is governed by the sender clock signal 12 (*CLK_put*). Each cell 370a-d communicates with the asynchronous get interface 372 to transmit data on the get data bus 56 (*get_data*), receives the global request for a get operation 54 (*get_req*), and each cell 370a indicates the end of the dequeuing operation on 55 (*get_ack*). Since the asynchronous get interface 50 only passes valid data (See FIG. 4), the valid bit is not used in the asynchronous get interface of cell 370a. Each cell receives the put token on put token

input 90 (*ptok_in*) and the get token on get token input 392 (*re1*); it passes the tokens on put token output 94 (*ptok_out*) and the get token on get token output 396 (*re*).

The synchronous part of cell 370a is identical to the corresponding components of cell 170 of FIFO circuit 100 (See FIG. 16). Referring to FIG. 30, the asynchronous part of cell 370a is decomposed into several blocks, e.g., ObtainGetToken block 393 (*OGT*) and asymmetric C-element 395. The ObtainGetToken block 393 (*OGT*) obtains the respective get token from the right interface on get token input 392 (*re1*). The ObtainGetToken block 393 (*OGT*) is implemented as a Burst-Mode asynchronous state machine as illustrated in FIG. 31(a). Another exemplary implementation of ObtainGetToken block 393' (*OGT*) is illustrated in FIG. 31(b). The Burst-Mode specification 420 is illustrated in FIG. 32 . The ObtainGetToken block 393 (*OGT*) observes the right cell and waits for a get operation. The right cell indicates a complete token passing by first setting and then resetting the get token input 392 (*re1*) (FIG. 30) at step 422 and 424, respectively. As those operations are completed, the get token is in the current cell, i.e., get token signal 397 (*gtok*) is set (step 424) in FIG. 32. The get token output 396 (*re*) (FIG. 30) is set at step 426 (FIG. 32), as controlled by the asymmetric C-element 395 (FIG. 30). It starts the get operation when the cell is full, when it has the get token and when the receiver request a data item. Once dequeuing is completed, communication with the receiver is finished by resetting the request and then the acknowledgment. The get token output 366 (*re*) is reset, along with the current cell's get token signal 397 (*gtok*) at 427.

Cell 370a performs a put operation in the same manner as cell 170a. When the cell is enabled on put enable signal 80 (*en_put*) and has the put token *ptok_in* = 1, the register 391 (*REG*) is enabled to enqueue data, as well as the put request 14 (*req_put*) which is used as the validity bit. In parallel, the data validity controller 380 (*DV_sa*) declares the cell 370 full. At the start of the clock cycle of sender clock 12 (*CLK_put*), data is latched into the register and the get token is passed to the next cell.

The get operation is performed as follows. Initially, the cell 370a starts without the get token (*gtok=0*). The ObtainGetToken block 393 (*OGT*) waits for an

up and down transition on the get token input 392 (*re1*); once these occur, get token is in the cell (*gtok* =1), and the output of the register 391 (*REG*) is driven onto the get data bus 56 (*get_data*). The latched validity bit is not used by the asynchronous get interface. The cell 370a waits for the receiver subsystem (or environment) to request a data item such that the get request signal 54 is asserted (*get_req* = 1). When this occurs, the cell 370a acknowledges it only if the cell contains a data item, i.e., full bit 388a is asserted (*f_i* = 1). When the three conditions are met, (i.e., *gtok*=1, *get_req*=1, and *f_i*=1), the get token output 396 (*re*) is set; this event acknowledges the data operation to the environment, starts resetting of the ObtainGetToken block 393 (*OGT*), starts resetting the cell's state in the data validity controller 380 (*DV_sa*), and starts sending the get token to the next cell. The operation cycle on the get interface is completed by de-asserting the get request signal 54 (*get_req*) which causes the de-assertion of get token output 396 (*re*) and the completion of all operations started on the positive edge of get token output 396 (*re*).

The data validity controller 380 (*DV_sa*) indicates when the cell is full or empty, and is illustrated in FIG. 34. The protocol for data validity controller 380 (*DV_sa*) is shown as a Petri-Net 430 in FIG. 35. In a normal empty state, the empty bit 382a is asserted (*e_i* = 1) and the full bit 388 is de-asserted (*f_i* = 0). When a put operation starts, data validity controller 380 (*DV_sa*) concurrently resets empty bit 382 (*e_i*) at step 431 and sets full bit 388 (*f_i*) at step 432 (i.e., the state of the cell 370a becomes "full"), thus enabling a get operation. The end of the put operation, i.e., de-assertion of write enable signal 394 (*we*) (step 433) can be performed concurrently with a get operation. A get operation is signaled by a pair of transitions on get token output 396 (*re*) (steps 434 and 435) after the falling transition on *re* occurs, the state of the cell changes to "empty" (*e_i* = 1 and *f_i* = 0) at steps 436 and 437, and the normal operation can resume.

Several additional embodiments described herein are substantially similar to FIFO circuits 100, 200, and 300, described above, but have been modified to operate as a relay station between a sender and a receiver. As illustrated in FIG. 36, a system 450 may include two subsystems, such as sender subsystem 452 and receiver subsystem 454, which are connected by very long wires 456 and 458. With

this configuration, a signal traveling between subsystem 452 and subsystem, 454 may take several clocks cycles to travel, and delay penalties may be introduced. As illustrated in FIG. 37, a modified system 460 introduces relay stations 462a, 462b, 462c, 462d to alleviate the connection delay penalties between subsystem 464 and subsystem 466, both of which must be operating under the same clock. The insertion of relay stations 462a, 462b, 462c, and 462d breaks the long wires into segments 468 and 470, each corresponding to less than one clock cycle delay. The chain of relay stations operate in a manner similar to FIFO circuits by sending packets from one system to another.

The implementation of a single-clock relay station, such as relay station 462b, as known in the art, is given in FIG. 38. Normally, the packets from the left relay station are passed to the right relay station.. In FIG. 37, packets are typically passed from relay station 462a to relay station 462b. The right relay station, i.e., relay station 462b, also has the capability to put counter-pressure on the data flow by stopping the relay stations to the left, i.e., relay station 462a. As illustrated in FIG. 38, exemplary relay station 462b is positioned between relay station 462a and relay station 462c (See FIG. 37). Referring to FIG. 38, relay station 462b has two registers. a main register 472 (*MR*) used in normal operation and an auxiliary register 474 (*AR*) used to store an extra packet when stopped. Unlike the mixed-clock cycle FIFO circuit 100, the relay station does not include full and empty signals, nor does it include put requests and get requests. Instead, in nominal operation, data is passed on every clock cycle. Since data passing is usually continuous, both valid and invalid data is transferred, where invalid data is passed whenever no valid data is enqueued. Hence, a valid bit is attached to both put and get data buses, forming a data packet on each interface.

With reference to FIGS. 37-38, relay station 462b (which is substantially identical to relay stations 462a, 462c, and 462d) operates as follows. In normal operation, at the beginning of every clock cycle, the data packet received on packet input signal 468 (*packetIn*) from the left relay station 462a is copied to main register 472 (*MR*) and then forwarded on packet output signal 470 (*packetOut*) to the right relay station 462c. A packet consists of a data item on a data bus and a valid bit

which indicates the validity of the data in the packet. If the receiver system 466 wants to stop receiving data, it asserts *stopIn* 476. On the next clock edge, the relay station 462b asserts *stopOut* 478, and latches the next packet to the auxiliary register 474 (*AR*). At this point, the cell will stall. When the relay station 462b is un-stalled, it will first send the packet from the main register 472 (*MR*) to the right relay station 462c, and subsequently the one from the auxiliary register 474 (*AR*).

Referring to FIG. 39, FIFO circuit 500 operates as a mixed-clock relay station. FIFO circuit 500 is substantially identical to FIFO circuit 100 (shown in FIGS. 5 and 6), with the differences noted herein. The interface of FIFO circuit 500 with the relay station is illustrated in FIG. 39. FIFO circuit 500 is placed in the chain of relay stations, such as that illustrated in FIG. 37, and interfaces between a left relay station chain, such as that including relay stations 462a and 462b, and a right relay station chain, such as that including relay stations 462c and 462d. Unlike system 460 illustrated in FIG. 37, which operates under a single clock, each relay station chain may be operated under a different clock, e.g., relay station chain 462a/462b operates under first clock domain 502, and relay station chain 462c/462d operates under second clock domain 504.

In contrast to FIFO circuit 100 described above, FIFO circuit 500 always passes valid data items from the left, put interface 506, to the right, get interface 508. In the protocol for FIFO circuit 500, there are no active requests on either interface. Instead, the get interface 508 and the put interface 506 are configured to actively stop, or interrupt, the continuous flow of data items. The get interface 508 dequeues data items from the FIFO circuit 500 on packet output signal 470 (*PacketOut*) on every clock cycle of receiver clock 22 (*CLK_get*). In order to stop the flow, relay station 462c asserts *stopIn* 476. Similarly, the FIFO circuit 500 always enqueues data items from the put interface 506 on packet input signal 468 (*packetIn*) . Thus, unlike FIFO circuit 100, put request signal 514 (*req_put*) is used solely to indicate data validity, and is treated as part of packet input signal 468 (*packetIn*) rather than a control signal. When it becomes full, FIFO circuit 500 stops the put interface 506 by asserting *stopOut* 512, which is the global full signal 18 (*full*). Thus unlike single clock relay station system 460, the mixed clock FIFO circuit relay

station 500 can be stalled on the put interface 506 and assert *StopOut* 478, even if no *StopIn* 476 has been asserted in the get interface 508.

FIFO circuit 500 is similar to FIFO circuit 100, with several modifications as noted herein. With respect to FIG. 40, the full detector 72 is implemented substantially as described above with respect to FIGS. 17-19 to determine the number of empty bits 582a, 582b, 58c, and 582d (*e_i*) in each of cells 570a, 570b, 570c, and 570d. The bi-modal empty detector 74 (See FIGS. 18-19) of FIFO circuit 100 is unnecessary, since the FIFO circuit relay station normally passes data on every clock cycle so deadlock cannot occur. Instead empty detector 574 is substantially identical to the near-empty detector 120 illustrated in FIG. 18, above. The put controller 176 and get controller 178 of FIFO circuit 100 which were described above with respect to FIGS. 20-21, are modified in FIFO circuit 500. In FIFO circuit 100, the put controller 176 enables the enqueuing of valid data items when it receives the put request signal 14 (*req_put*). Put controller 576 of FIFO circuit 500 simply allows valid data items to pass through. Put controller 576 continuously enqueues data items unless the FIFO circuit 500 becomes full. Thus, the put controller 576 is implemented in the exemplary embodiment as an inverter on the global full signal 18 (*full*) (See FIG. 41). In contrast to get controller 178 of FIFO circuit 100, in which dequeuing was done on demand, get controller 578 enables continuous dequeuing of data items. As illustrated in FIG. 42, dequeuing is interrupted only when FIFO circuit 500 becomes empty (the near empty signal 122 (*ne*) is asserted, as described above) or the get interface signals it can no longer accept data items by asserting *stopIn* 476. Since both valid and invalid data may be passed, get controller 678 also uses the valid bit 584 (*valid_i*) to compute the validity signal 30 (*valid_get*).

Referring to FIGS. 43, 44, 45, 47, 48 and 49, FIFO circuits 600 and 700 are two additional embodiments that operate as relay stations which are configured to operate with mixed asynch-synch and synch-asynch interfaces, respectively. FIFO circuits 600 and 700 simultaneously address two critical design challenges: the capability of interfacing between mixed asynch/synch environments and long inter-connect delays.

The basic architecture of communication between an asynchronous sender 490 and a synchronous receiver 466 with relay stations is illustrated in FIG. 43. The asynchronous domain sends data packets (possibly through a chain of asynchronous relay stations (*ARS*) 494a and 494b, discussed in greater detail herein) to FIFO circuit 600. The packets are then transferred to the synchronous domain, and sent through the chain of synchronous relay stations 462a and 462b to the receiver 466.

In principle, communication at the asynchronous interface can be made arbitrarily robust, so that no relay stations are needed at the asynchronous domains outputs. In practice, however, correctness and performance issues need to be addressed in FIFO designs. Two common asynchronous data encoding styles are known in the art: dual-rail and single-rail bundled data. (Single-rail bundled data is described in greater detail in S. Furber, "Asynchronous Design," *Proc. of Submicron Electronics*, pp. 461-492, 1997, which is incorporated by reference in its entirety herein.) The dual-rail style, which encodes both the value and the validity of each data bit on a pair of wires, is arbitrarily robust with respect to wire delays (but has significant overhead) and does not require a chain of *ARS*'s. The single-rail bundled-data style has timing assumptions between the data itself and the control wires, so a chain of *ARS*'s may be desirable to limit the wire lengths between stages to short hops. Finally, for the issue of performance, even if *ARS*'s are not required, they may he desirable to increase the throughput. A chain of *ARS*'s can be directly implemented by using an asynchronous FIFO circuit commonly known as a *micro-pipeline* (Further details about micropipelines are described in I Sutherland, "Micropipelines," *Communications of the ACM*, 32(6), pp. 720-738, 1989 and M. Singh et al., MOUSETRAP: Ultra High-Speed Transition-Signaling Asynchronous Pipelines," *ACM TAU-00 Workshop*, 2000, both of which are incorporated by reference in their entirety herein.)

Unlike the synchronous data packets, the asynchronous data packets do not need a validity bit. Rather, the presence of valid data packets is signaled on the control request wires and an *ARS* can wait indefinitely between receiving data packets. Therefore, a standard micropipeline implements the desired *ARS* behavior.

FIFO circuit 600, which is illustrated in FIGS. 44 and 45, operates between an asynchronous domain 602 and a synchronous domain 604. The asynchronous interface 640 is identical and supports the same communication protocol with the asynchronous interface 40 in FIFO circuit 200 previously described. This interface matches exactly the micropipeline interfaces. Similarly, the synchronous interface 620 is identical and supports the same communication protocol with the respective synchronous get interface 508 in FIFO circuit 500. Referring to FIG. 45, at the architectural level, the FIFO circuit 600 is identical to FIFO circuit 200 shown in FIG. 8. Cells 670a, 670b, 670c, and 670d are substantially identical to cells 270a, 270b, 270c, and 270d. The get controller 678 is different from get controller 278 (which is substantially identical to get controller 178), as will be described below. Empty detector 674 is substantially identical to empty detector 574 in FIFO circuit 500, and corresponds to the near empty detector 120 illustrated in FIG. 18, above.

FIFO circuit 600 operates as follows. Whenever a data item is present at its asynch interface 640, FIFO circuit 600 enqueues it. On the synchronous interface 620, FIFO circuit 600 outputs a data item every clock cycle unless it is empty or it is stopped by the right relay station. Thus, unlike FIFO circuit 500, a data packet is invalid only if the FIFO circuit 600 is stalled. The get interface 620 is stalled when the FIFO circuit 600 is empty or stopped from the right. However, since the FIFO circuit 600 does not enqueue invalid data packets, the right interface receives only valid data packets unless the FIFO circuit 600 is stalled.

The implementation of the get controller 678 is illustrated in FIG. 46. The get controller 678 enables a get operation (*en_get* = 1) when it is not stopped from the right (*stopIn* = 0) and when the relay station is not empty (*ne* = 0). This operates in the same manner as get controller 178 of FIFO circuit 100. The packet sent to the right is invalid if either the relay station is stopped or it is empty. Therefore, all the packets received from the asynchronous interface are valid, and, thus, there is no need for an distinct validity bit, instead the get enable signal 686 (*en_get*) is used as the validity signal 30 (*valid_get*).

The basic architecture of communication between a synchronous sender 464 and an asynchronous receiver 492 with relay stations is illustrated in FIG.

47. The synchronous domain sends data packets through a chain of synchronous relay stations 462a and 462b to FIFO circuit 800. The packets are then transferred to the asynchronous receiver 292, preferably through the chain of *ARS* 494a and 494b in the asynchronous domain.

The interfaces of FIFO circuit 700 are illustrated in FIG. 48. The asynchronous interface 750 is substantially identical to the asynchronous interface 50 of FIFO circuit 500 shown in FIG. 9, and supports the same communication protocol. Similarly, the synchronous interface 710 shown in FIG. 48 is substantially identical to synchronous interface 10 of FIFO circuit 300 and supports the same communication protocol. Cells 770a, 770b, 770c, and 770d of the FIFO circuit 700 shown in FIG. 49 are substantially identical to cells 370a, 370b, 370c, and 370d of the FIFO circuit shown in FIG. 10. The only significant difference between FIFO circuit 700 and FIFO circuit 300 is the put controller of each FIFO circuit.

During normal operation, FIFO circuit 700 transmits data packets from the synchronous interface to the asynchronous one. The asynchronous relay stations on the right enqueue a data packet whenever the FIFO circuit 700 supplies data. However, on the synchronous interface, the FIFO circuit 700 acts as a filter since all asynchronous data packets must be valid. The validity bit 14 (*valid_put*) of the incoming synchronous packets is used to filter them. More particularly, when the packet is valid, FIFO circuit 700 is configured to enqueue it; otherwise it is configured to discard it. FIFO circuit 700 enqueues only valid data packets and stalls the put interface when the FIFO circuit 700 is full. FIFO circuit 500, as described above, enqueues all data packets received, and stalls when there are no more full cells. In contrast, FIFO circuit 700 stalls under the following conditions: (1) when the FIFO is full and/or (2) when an invalid data packet is received.

The exemplary implementation of the put controller 776 is given in FIG. 50. The put controller 776 enables a put operation (*en_put* = 1) only when the relay station is not full (*full* = 0) and the incoming data packet is valid (*valid_put* = 1). The put controller 776 implementation is similar to put controller 176, but the role of the explicit put request signal (such as *req_put*) has been taken by an implicit valid bit 14 (*valid_put*) which accompanies every packet.

Example

In order to evaluate the performance of the various FIFO circuit designs, Each of the exemplary FIFO circuit 100, 200, 300, 500, 600, and 700 were simulated. Each FIFO circuit was simulated using both commercial and academic tools. The designs were built using both library and custom circuits, and were simulated using Cadence HSPICE. The Burst-Mode controllers were synthesized using Minimalist (Minimalist is described in greater detail in R. Fuhrer et al., "MINIMALIST: An Environment for Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines," *CUCS-020-99,* 1999, which is incorporated by reference in its entirety herein.) and the Petri-Net controllers were synthesized using Petrify (Petrify is described in greater detail in J. Cortadella et al., "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," *IEICE Transactions on Information and Systems,* Vol. E80-D, Number 3, pp. 315-325, March 1997, which is incorporated by reference in its entirety herein). The FIFO circuit designs were simulated in 0.6μ HP CMOS technology, at 3.3V and 300K.

The following are among the special considerations in modeling the control and data global buses: The control buses *put_req/en_put*, and *get_req/en_get* were inserted with appropriate buffering. The acknowledgement signal *put_ack* and *get_ack* are constructed as a tree of OR gates that merge individual acknowledgements into a single global one. In modeling *get_data* and *data_get*, each bit in the bus is driven by tri-state buffers. Both the load contributed by the environment and by the long wires within the FIFO circuit were modeled. The model made the assumption that the environment contributes to the load with two inverters (roughly corresponding to a latch), and that each wire contributes with a capacitance of two inverters per cell (roughly *2n* inverters per wire).

Two metrics have been simulated for each design: *latency* and *throughput*. Latency is the delay from the input of data on the put interface, to its presence at the output on the get interface in an empty FIFO circuit. Throughput is defined as the reverse of the cycle time for a put or get operation. The throughput and latency have been computed for different FIFO circuit capacities and data item

widths. The FIFO circuit's capacity has been set to four, eight, or 16 cells. For each of these FIFO circuit's, the data item width has been set to either eight or 16 bits.

The results for maximum throughput are given in TABLES 1 and 2. For synchronous interfaces, the throughput is expressed as the maximum clock frequency with which that interface can be clocked. Since the asynchronous interfaces do not have a clock, the throughput is given in MegaOps/s (the number of data operations the interface can perform in a second).

Embodiment	4-place		8-place		16-place	
	put	get	put	get	put	get
Circuit 100	565	549	544	523	505	484
Circuit 200	421	549	379	523	357	484
Circuit 300	565	454	544	392	505	360
Circuit 500	580	539	550	517	509	475
Circuit 600	421	539	379	517	357	475
Circuit 700	580	454	550	392	509	360

TABLE 1

Embodiment	4-place		8-place		16-place	
	put	get	put	get	put	get
Circuit 100	505	492	488	471	460	439
Circuit 200	386	492	351	471	332	439
Circuit 300	505	417	488	362	460	335
Circuit 500	521	478	498	459	467	430
Circuit 600	386	478	351	459	332	430
Circuit 700	521	417	498	392	467	360

TABLE 2

The throughput results are consistent with the FIFO circuit designs. The synchronous get interfaces are slower than the synchronous put interface because of the complexity of the empty detector 74. Also, relay-stations synchronous put interfaces are somewhat faster than their FIFO circuit's counterparts due to the simplification of put detector in the former ones. On the synchronous side, the get

interface tends to be faster than the put one mainly because the output of the register is enabled early on the data bus.

Latencies (ns) through empty FIFO circuit's are shown only for designs with 8 bit data items (TABLE 3). The experimental setup for latency is as follows: in empty FIFO

Embodiment	4-place		8-place		16-place	
	Min	Max	Min	Max	Min	Max
Circuit 100	5.43	6.34	5.79	6.64	6.14	7.17
Circuit 200	5.53	6.45	6.13	7.17	6.47	7.51
Circuit 300	1.95		2.18		2.44	
Circuit 500	5.48	6.41	6.05	7.02	6.23	7.28
Circuit 600	5.61	6.35	6.18	7.13	6.57	7.62
Circuit 700	1.86		2.23		2.43	

TABLE 3

circuit, the get interface requests a data item. At an arbitrary time later, the put interface places a data item. the latency is computed as the elapsed time between the moment when the put data bus has valid data to the moment when the get interface retrieves the data item and can use it.

Latency for a FIFO circuit with a synchronous receiver is not uniquely defined. Latency varies with the moment when data items are safely enqueued in a cell. If the data item is enqueued by the put interface immediately after the positive edge of *CLK_get*, latency is increased (column *Max* in the table). If the data item is enqueued right before the empty detector starts computation, then latency is decreased (column *Min*). However, an asynchronous receiver is able to grab the data item immediately after its enqueuing; therefore, latency is uniquely defined. More interestingly, since the asynchronous receiver does not need synchronization, the latency for communication to an asynchronous domain can be performed much faster.

The throughput and latency results are quite good for a bus-based design. As expected, both the throughput and latency decrease when the FIFO circuit capacity or the data items width increase. The throughput tends to be higher for the

synchronous interfaces than for the asynchronous ones. The latency through an empty FIFO circuit is smaller when the receiver is asynchronous.

It will be understood that the foregoing is only illustrative of the principles of the invention, and that various modifications can be made by those skilled in the art without departing from the scope and spirit of the invention.

CLAIMS

1. A FIFO circuit which interfaces the transmission of data items between a sender subsystem operating under a first time domain and a receiver subsystem operating under a second time domain, wherein the first time domain and the second time domain are different and at least one of the time domains operates according to a clock signal, the FIFO circuit comprising:

a put interface configured to operate according to the first time domain comprising a put data bus to transmit a data item from the sender subsystem and a put data request input to receive a put request from the sender subsystem to enqueue the data item from the put data bus;

a get interface configured to operate according to the second time domain comprising a get data bus to transmit the data item to the receiver subsystem and a get data request input to receive a get request from the sender subsystem to dequeue the data item to the get data bus; and

an array of cells, each cell comprising:

a register configured to receive the data item from the put data bus and to transmit the data item to the get data bus;

a state controller providing an indication of the state of the cell;

a put component configured to operate according to the first time domain to receive a put token from a first adjacent cell, to latch the data item received from the put data bus to the register based on the put request, the put token, and the state of the cell, and to pass the put token to a second adjacent cell; and

a get component configured to operate according to the second time domain to receive the get token from the first adjacent cell, to dequeue the data item from the register to the get data bus based on the get request, the get token, and the state of the cell, and pass the get token to the second adjacent cell.

2. The FIFO circuit of claim 1, wherein the put interface is synchronous and controlled by a first clock signal, and the put interface further comprises:

a full detector which produces a global full signal synchronized with the first clock signal which is asserted when fewer than a predetermined number of consecutive cells in the array of cells are in an empty state; and

a put controller configured to deliver the put request to the put component of each cell in the array of cells when the global full signal is not asserted.

3. The FIFO circuit of claim 2, wherein the put component of each cell in the array of cells further comprises:

a synchronous latch enabled by the put request to pass the put token to the second adjacent cell;

a circuit configured to signal the state controller to provide an indication de-asserting an empty state of the cell in response to the put request and the put token.

4. The FIFO circuit of claim 2, wherein the register is enabled by the put request to receive the data item.

5. The FIFO circuit of claim 2, wherein the get interface is synchronous and controlled by a second clock signal, and the get interface further comprises:

a first empty detector which produces a first global empty signal synchronized with the second clock signal which is asserted when fewer than a predetermined number of consecutive cells in the array of cells are in a full state;

a second empty detector which produces a second global empty signal and which comprises: a circuit which produces a first intermediate signal that is asserted when none of the cells in the array of cells are in the full state; a first and second latch which synchronize the first intermediate signal with the second clock signal ; and a combination element which combines a second intermediate signal with the first intermediate signal between first latch and the second latch, wherein the second global empty signal is the first intermediate signal when the second intermediate signal is de-asserted and the second global empty signal is the second intermediate signal when the second intermediate signal is asserted; and

a get controller configured to deliver the get request to the get component of each cell in the array of cells when one the first global empty signal and the second global empty signal is de-asserted.

6. The FIFO circuit of claim 5, wherein the get component of each cell in the array of cells further comprises:

a synchronous latch enabled by the get request to pass the get token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication de-asserting a full state of the cell in response to the get request and the get token.

7. The FIFO circuit of claim 5, wherein the register is enabled by the get request and the get token to transmit the data item to the get data bus .

8. The FIFO circuit of claim 2, wherein the get interface is asynchronous, and the get component of each cell in the array of cells comprises a get token passing circuit configured to receive the get token in the respective cell in the array of cells in response to a signal from the first adjacent cell indicative of dequeueing the data item from the register of the first adjacent cell to the get data bus.

9. The FIFO circuit of claim 8, wherein the register is enabled by the get token to dequeue the data item to the get data bus.

10. The FIFO circuit of claim 9, wherein the get interface further comprises a get acknowledgement signal and the get component further comprises a get control circuit enabled by the get token, the get request, and the full state of the cell to signal the state controller to reset the state of the cell, to assert the get acknowledgement signal.

11. The FIFO circuit of claim 10, wherein the get control circuit is disabled by de-assertion of the get request to pass the get token to the second adjacent cell.

12. The FIFO circuit of claim 1, wherein the put interface is asynchronous, and the put component of each cell in the array of cells comprises a put token passing circuit configured to receive the put token into the respective cell in the array of cells in response to a signal from the first adjacent cell indicative of latching the data item from the put data bus to the register of the first adjacent cell.

13. The FIFO circuit of claim 12, wherein the put interface further provides a put acknowledgement signal and the put component further comprises a put control circuit enabled by the put token, the put request, and the empty state of the cell to enable the register to receive the data item from the put data bus, to signal the state controller to provide an indication of the resetting of the state of the cell, and to assert the put acknowledgement signal.

14. The FIFO circuit of claim 12, wherein the put control circuit is disabled by de-assertion of the put request to pass the put token to the second adjacent cell.

15. The FIFO circuit of claim 12, wherein the second time domain is synchronous and controlled by a second clock signal, and the get interface further comprises:

a first empty detector which produces a first global empty signal synchronized with the second clock signal which is asserted when fewer than a predetermined number of consecutive cells in the array of cells are in the full state;

a second empty detector which produces a second global empty signal and which comprises: a circuit which produces a first intermediate signal that is asserted when none of the cells in the array of cells are in the full state; a first and second latch which synchronize the first intermediate signal with the second clock signal ; and a combination element which combines a second intermediate signal with the first intermediate signal between first latch and the second latch, wherein the second global empty signal is the first intermediate signal when the second intermediate signal is de-asserted and the second global empty signal is the second intermediate signal when the second intermediate signal is asserted; and

a get controller configured to deliver the get request to the get component of each cell in the array of cells when one the first global empty signal and the second global empty signal is de-asserted.

16. The FIFO circuit of claim 15, wherein the get component of each cell in the array of cells further comprises:

a synchronous latch enabled by the get request to pass the get token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication of the empty state of the cell in response to the get request and the get token.

17. The FIFO circuit of claim 16, wherein the register is enabled by the get request and the get token to transmit the data item to the get data bus.

18. A circuit which interfaces the transmission of a data item from a sender subsystem controlled by a first clock signal to a receiver subsystem controlled by a second clock signal wherein the transmission of the data item is subject to a delay between the sender subsystem and the receiver subsystem, the circuit comprising:

a first chain of relay stations attached to the sender subsystem to transmit the data item on a put data bus and having a first protocol of operation;

a second chain of relay stations attached to the receiver subsystem to receive the data item on a get data bus and having a second protocol of operation; and

a mixed clock relay station which receives the first clock signal and the second clock signal and transmits the data item from the first chain of relay stations to the second chain of relay stations in accordance with the protocol of operation of the first chain of relay stations and the protocol of operation of the second chain of relay stations, wherein the mixed clock relay station comprises:

an array of cells;

a full detector which produces a full signal synchronized with the first clock signal which is asserted when fewer than a predetermined number of consecutive cells in the array of cells are in the empty state;

a put controller configured to enable the enqueuing of a data item on each clock cycle of the first clock signal if a full signal is not asserted;

an empty detector which produces an empty control signal synchronized with the second clock signal which is asserted when fewer than a predetermined number of consecutive cells are in the full state; and

a get controller configured to receive a stop signal from a relay station of the second chain of relay stations connected to the mixed clock station, and configured to enable the dequeuing of a data item on each clock cycle of the second clock signal if the empty signal is not asserted and the stop signal is not asserted.

19. The circuit of claim 18, wherein each cell in the array of cells comprises:

a register configured to receive the data item from the put data bus and to transmit the data item to the get data bus;

a state controller providing an indication of the state of the cell;

a put component configured to operate according to the first protocol of operation to receive a put token from a first adjacent cell, to latch the data item received from the put data bus to the register based on the put request, the put token, and the state of the cell, and to pass the put token to a second adjacent cell; and

a get component configured to operate according to the second protocol of operation to receive a get token from the first adjacent cell, to dequeue the data item from the register to the get data bus based on the get request, the get token, and the state of the cell, and pass the get token to the second adjacent cell.

20. The circuit of claim 19, wherein a relay station of the first chain of relay stations chain provides a put request and the put component of each cell in the array of cells further comprises:

a synchronous latch enabled by the put request to pass the put token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication of the de-assertion of an empty state of the cell in response to the put request and the put token.

21. The circuit of claim 20, wherein the register is enabled by the put request and the put token to receive the data item.

22. The circuit of claim 21, wherein a relay station of the second chain of relay stations chain provides a get request and the get component of each cell in the array of cells further comprises:

a synchronous latch enabled by the get request to pass the get token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication of the de-assertion of the full state of the cell in response to the get request and the get token.

23. The FIFO circuit of claim 22, wherein the register is enabled by the get request and the get token to transmit the data item to the get data bus.

24. A circuit which interfaces the transmission of a data item from an asynchronous sender subsystem to a synchronous receiver subsystem controlled by a clock signal wherein the transmission of the data item is subject to a delay between the sender subsystem and the receiver subsystem, the circuit comprising:

a first chain of relay stations attached to the sender subsystem to transmit the data item on a put data bus and having an asynchronous protocol of operation;

a second chain of relay stations attached to the receiver subsystem to receive a data item on a get data bus and having a synchronous protocol of operation; and

a relay station which receives the clock signal and transmits the data item from the first chain of relay stations to the second chain of relay stations in accordance with the asynchronous protocol of operation of the first chain of relay stations and the synchronous protocol of operation of the second chain of relay stations, wherein the relay station comprises:

an array of cells;

an empty detector which produces an empty control signal synchronized with the second clock signal which is asserted when fewer than a predetermined number of consecutive cells are in the full state; and

a get controller configured to receive a stop signal from a relay station of the second chain of relay stations connected to the mixed clock station, and configured to enable the dequeuing of a data item on each clock cycle of the second clock signal if the empty signal is not asserted and the stop signal is not asserted.

25. The circuit of claim 24, wherein each cell in the array of cells comprises:

a register configured to receive the data item from the put data bus and to transmit the data item to the get data bus;

a state controller providing an indication of the state of the cell;

a put component configured to operate according to the first protocol of operation to receive a put token from a first adjacent cell, to latch the data item

received from the put data bus to the register based on the put request, the put token, and the state of the cell, and to pass the put token to a second adjacent cell; and

a get component configured to operate according to the second protocol of operation to receive a get token from the first adjacent cell, to dequeue the data item from the register to the get data bus based on the get request, the get token, and the state of the cell, and pass the get token to the second adjacent cell.

26. The circuit of claim 25, wherein the put component of each cell in the array of cells comprises a put token passing circuit configured to receive the put token into the respective cell in the array of cells in response to a signal from the first adjacent cell indicative of latching the data item from the put data bus to the register of the first adjacent cell.

27. The circuit of claim 26, wherein a relay station of the first chain of relay stations chain provides a put request and the put interface further provides a put acknowledgement signal and the put component further comprises a put control circuit enabled by the put token, the put request, and the empty state of the cell to enable the register to receive the data item from the put data bus, to signal the state controller to provide an indication of the resetting of the state of the cell, and to assert the put acknowledgement signal.

28. The circuit of claim 26, wherein the put control circuit is disabled by de-assertion of the put request to pass the put token the second adjacent cell.

29. The circuit of claim 28, wherein a relay station of the second chain of relay stations chain provides a get request and the get component of each cell in the array of cells further comprises:

a synchronous latch enabled by the get request to pass the get token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication of the empty state of the cell in response to the get request and the get token.

30. The circuit of claim 29, wherein the register is enabled by the get request and the get token to transmit the data item to the get data bus.

31. A circuit which interfaces the transmission of a data item from a synchronous sender subsystem controlled by a clock signal to an asynchronous receiver subsystem wherein the transmission of the data item is subject to a delay between the sender subsystem and the receiver subsystem, the circuit comprising:

a first chain of relay stations attached to the sender subsystem to transmit the data item on a put data bus and having a synchronous protocol of operation and providing a validity signal;

a second chain of relay stations attached to the receiver subsystem to receive the data item on a get data bus and having an asynchronous protocol of operation; and

a relay station which receives the clock signal and transmits the data item from the first chain of relay stations to the second chain of relay stations in accordance with the synchronous protocol of operation of the first chain of relay stations and the asynchronous protocol of operation of the second chain of relay stations, wherein the relay station comprises:

an array of cells;

a full detector which produces a full signal synchronized with the first clock signal which is asserted when fewer than a predetermined number of consecutive cells in the array of cells are in an empty state;

a put controller configured to enable the enqueuing of a data item on each clock cycle of the first clock signal if a full signal is not asserted and the validity signal is asserted.

32. The circuit of claim 31, wherein each cell in the array of cells comprises:

a register configured to receive the data item from the put data bus and to transmit the data item to the get data bus;

a state controller providing an indication of the state of the cell;

a put component configured to operate according to the first protocol of operation to receive the put token from a first adjacent cell, to latch the data item

received from the put data bus to the register based on the put request, the put token, and the state of the cell, and to pass the put token to a second adjacent cell; and

a get component configured to operate according to the second protocol of operation to receive the get token from the first adjacent cell, to dequeue the data item from the register to the get data bus based on the get request, the get token, and the state of the cell, and to pass the get token to the second adjacent cell.

33. The circuit of claim 32, wherein a relay station of the first chain of relay stations chain provides a put request and the put component of each cell in the array of cells further comprises:

a synchronous latch enabled by the put request to pass the put token to the second adjacent cell; and

a circuit configured to signal the state controller to provide an indication of the empty state of the cell in response to the put request and the put token.

34. The circuit of claim 33, wherein the register is enabled by the put request to receive the data item.

35. The FIFO circuit of claim 34, the get component of each cell in the array of cells comprises a get token passing circuit configured to receive the get token in the respective cell in the array of cells in response to a signal from the first adjacent cell indicative of dequeueing the data item from the register of the first adjacent cell to the get data bus.

36. The FIFO circuit of claim 35, wherein the register is enabled by the get token to dequeue the data item to the get data bus.

37. The FIFO circuit of claim 36, wherein a relay station of the second chain of relay stations chain provides a get request and the get interface further comprises a get acknowledgement signal and the get component further comprises a get control circuit enabled by the get token, the get request, and the full state of the cell to signal the state controller to provide an indication of the resetting of the state of the cell, to assert the get acknowledgement signal.

38. The FIFO circuit of claim 37, wherein the get control circuit is disabled by de-assertion of the get request to pass the get token to the second adjacent cell.

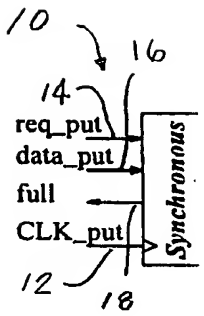


FIG. 1

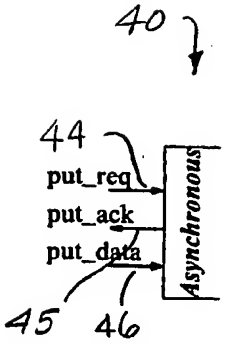


FIG. 3

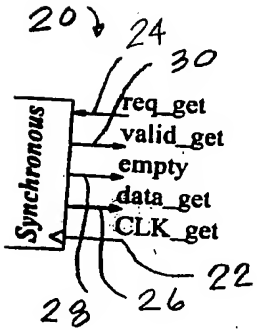


FIG. 2

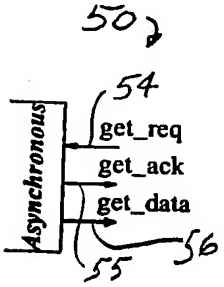


FIG. 4

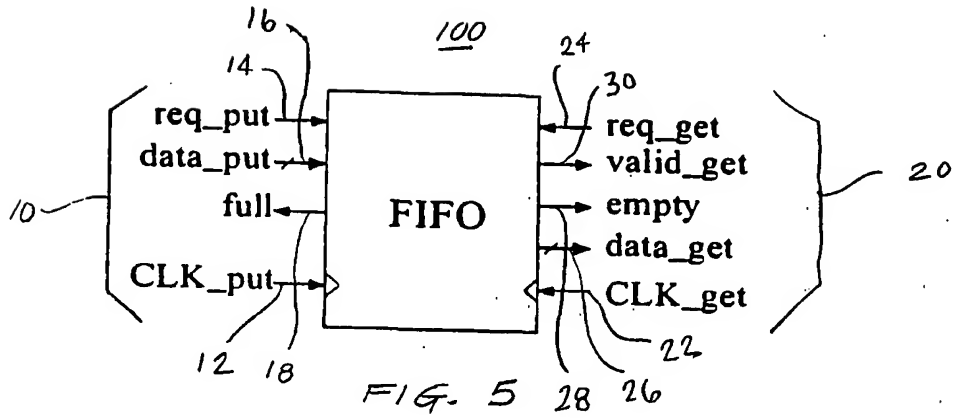

16
100
24
14
30
req_put
req_get
data_put
valid_get
10
full
FIFO
empty
20
data_get
CLK_put
CLK_get
22
12
18
28
26

FIG. 5

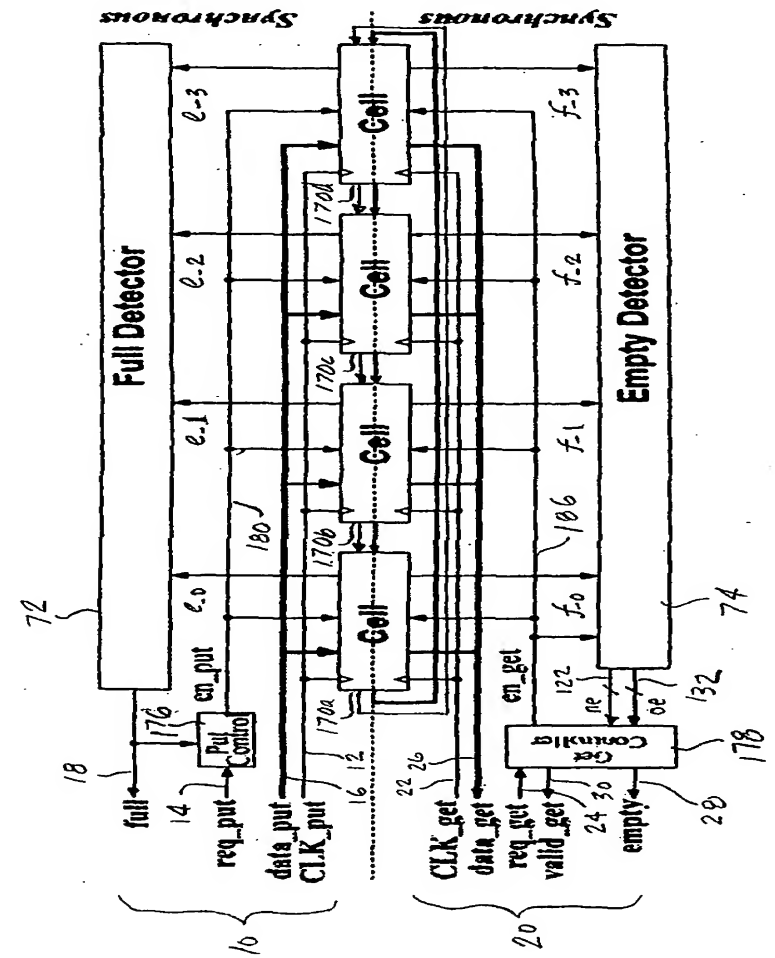

Synchronous
Synchronous
Full Detector
Empty Detector
Cell
Cell
Cell
Cell
e-3
e-2
e-1
e-0
f-3
f-2
f-1
f-0
170d
170c
170b
170a
180
186
172
174
176
178
en_put
en_get
Ful Control
full
req_put
data_put
CLK_put
CLK_get
data_get
req_get
valid_get
empty
ne
oe
18
14
16
12
22
26
24
30
28
132
122
10
20
100

FIG. 6

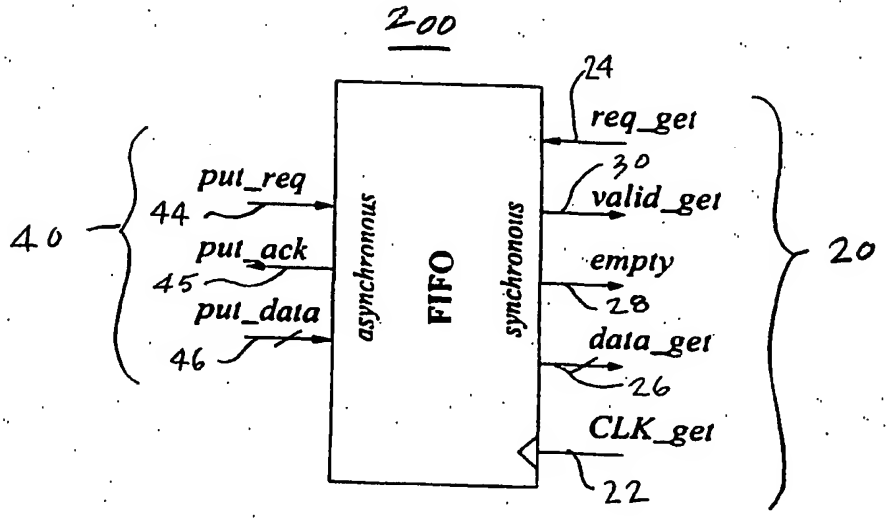

200
24
req_get
30
valid_get
empty
28
data_get
26
CLK_get
22
20
40
put_req
44
put_ack
45
put_data
46
asynchronous
FIFO
synchronous

FIG. 7

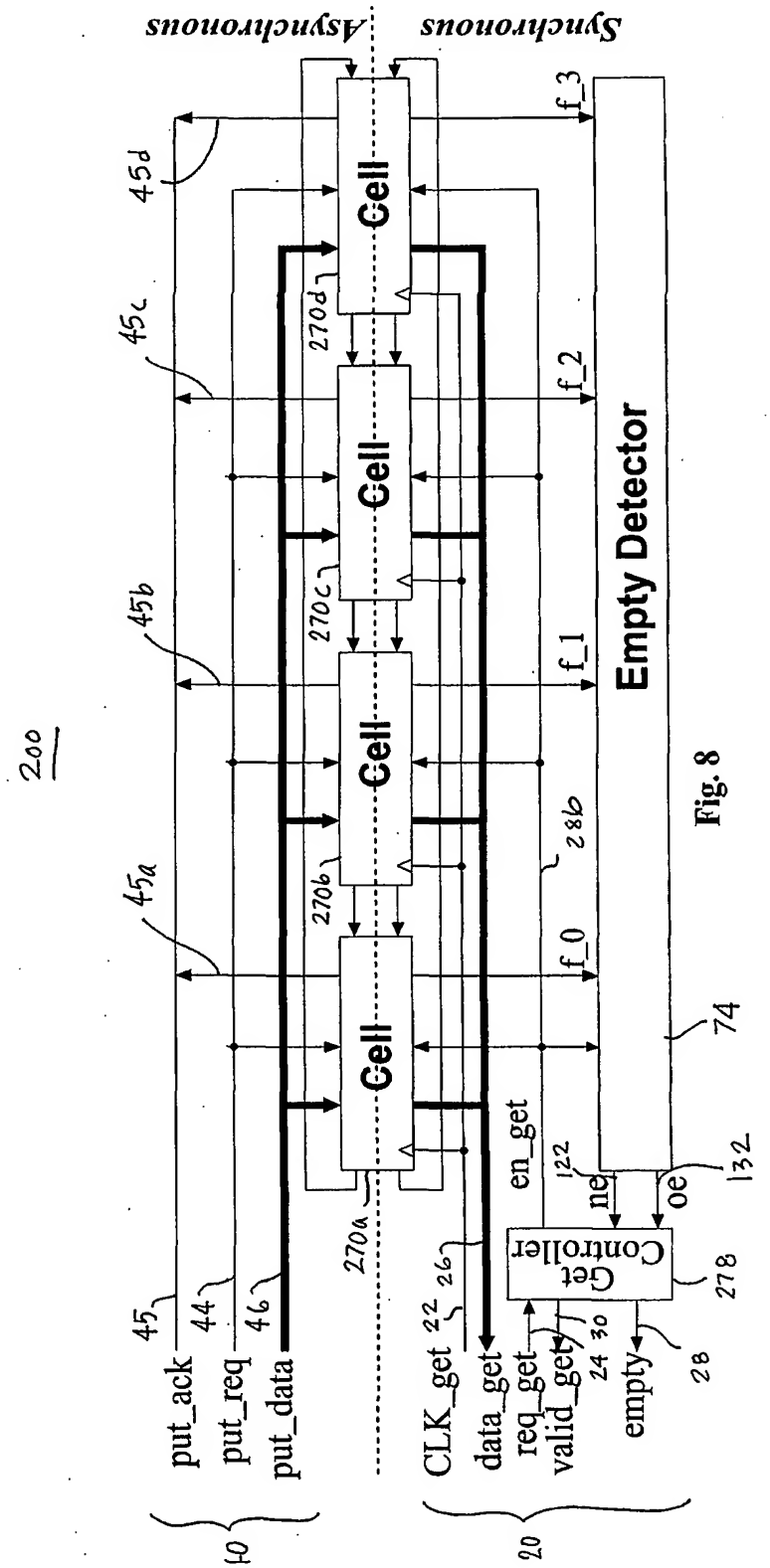

Asynchronous
Synchronous
200
Cell
Cell
Cell
Cell
Empty Detector
Get Controller
45d
45c
45b
45a
270d
270c
270b
270a
f_3
f_2
f_1
f_0
286
74
122
132
278
45
44
46
26
22
24
30
28
put_ack
put_req
put_data
CLK_get
data_get
req_get
valid_get
empty
en_get
ne
oe
10
20

Fig. 8

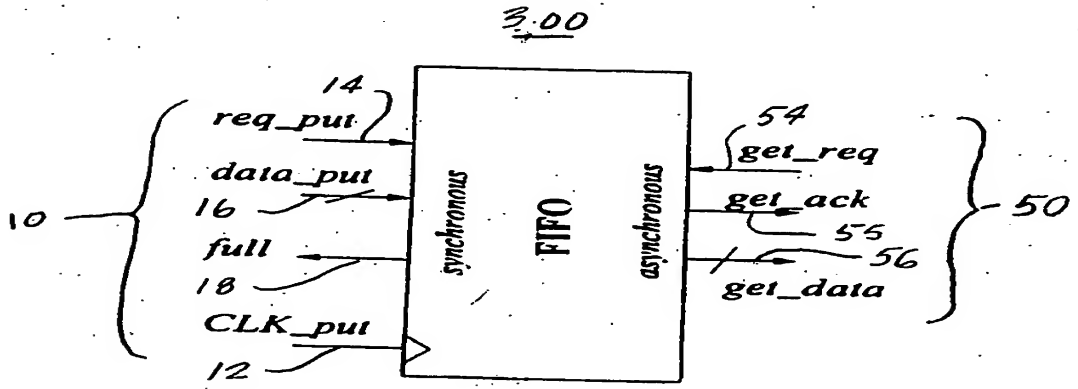

300
14
req_put
data_put
16
full
18
CLK_put
12
10
synchronous
FIFO
asynchronous
54
get_req
get_ack
56
56
get_data
50

FIG. 9

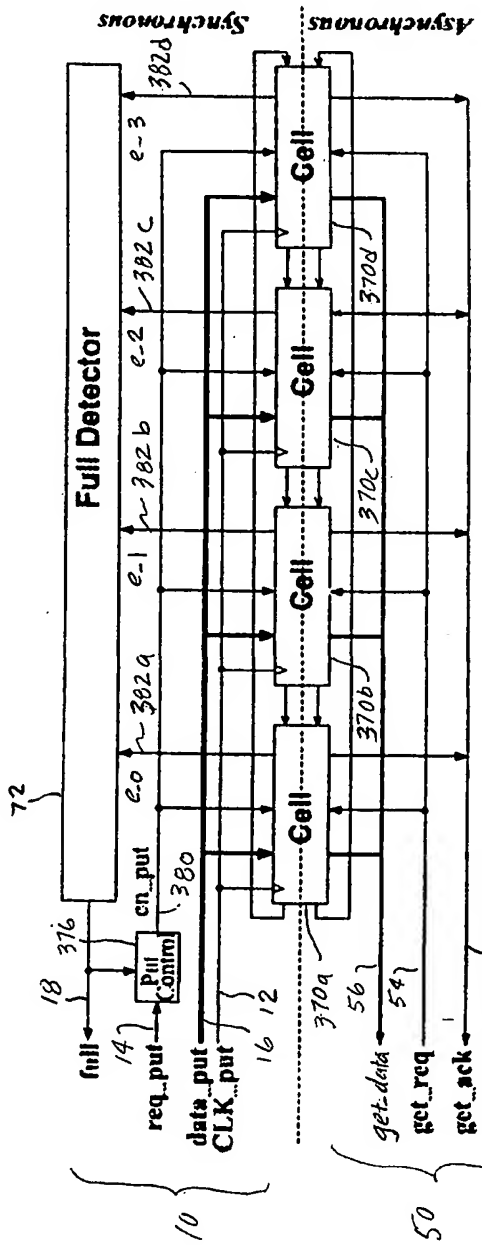

Synchronous
Asynchronous
Full Detector
Cell
Cell
Cell
Cell
e-3
e-2
e-1
e-0
382d
382c
382b
382a
370d
370c
370b
370a
72
300
cn_put
380
376
18
full
14
req_put
data_put
CLK_put
16
12
56
54
55
get_data
get_req
get_ack
10
50

FIG. 10

FIG. 11

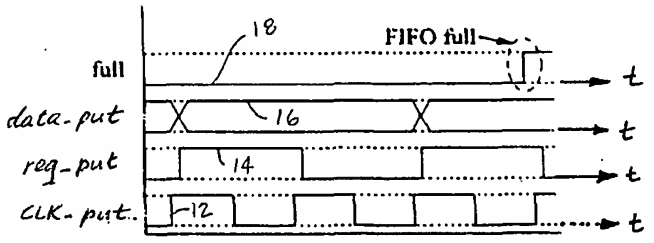


FIG. 12

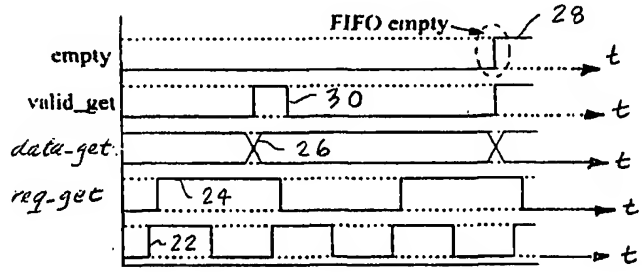


FIG. 13

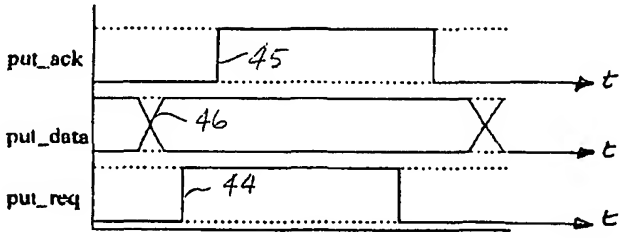
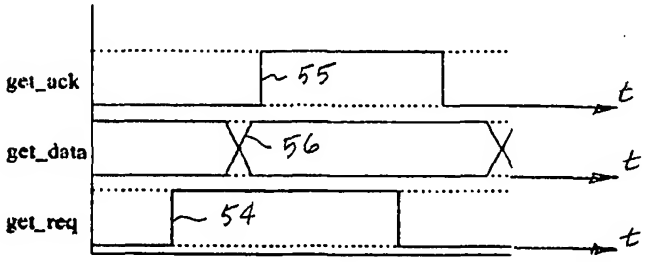


FIG. 14



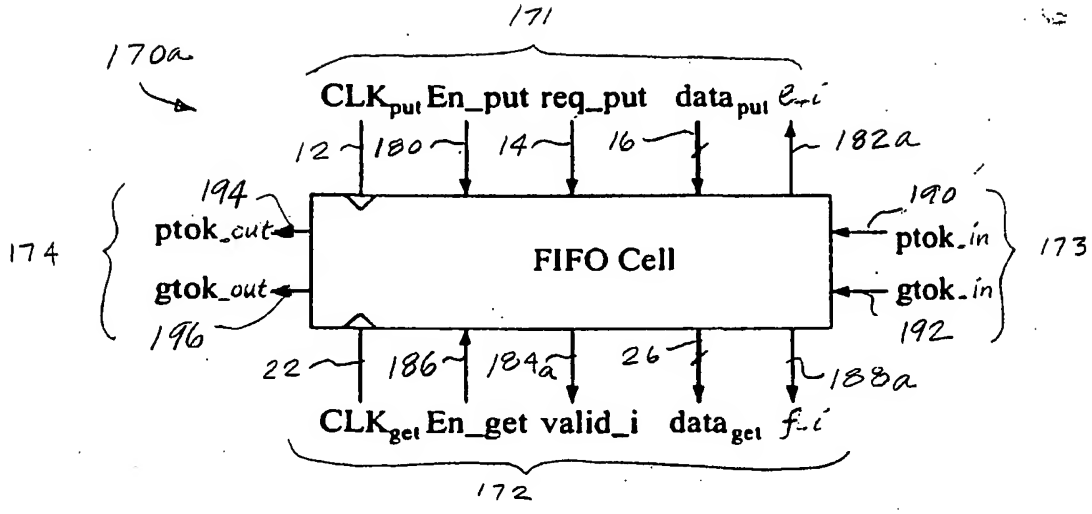

171
170a
CLK_put En_put req_put data_put e_i
12
180
14
16
182a
194
190
174
ptok_out
ptok_in
173
FIFO Cell
gtok_out
gtok_in
196
192
22
186
184a
26
188a
CLK_get En_get valid_i data_get f_i
172

FIG. 15

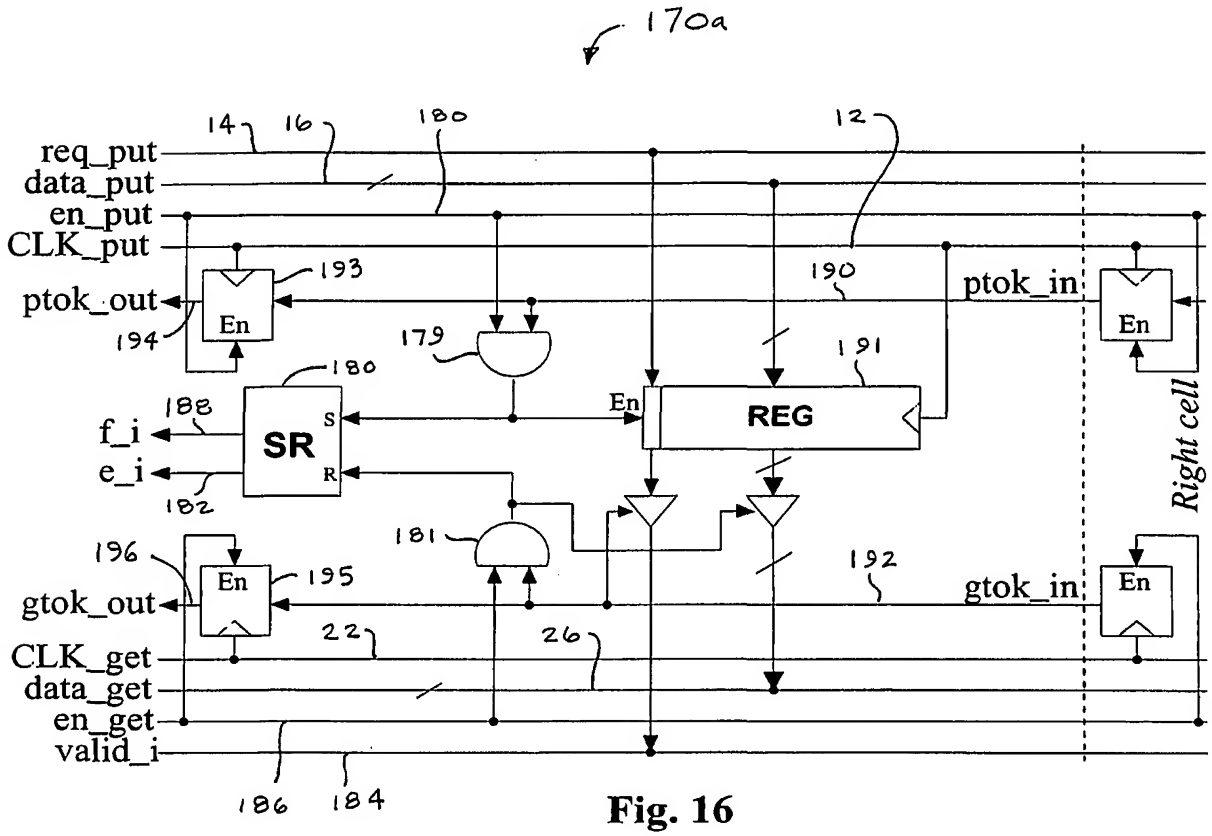

170a
14
16
180
12
req_put
data_put
en_put
CLK_put
ptok_out
193
194
En
179
190
ptok_in
191
180
188
f_i
e_i
182
SR
S
R
En
REG
Right cell
181
196
195
gtok_out
192
gtok_in
22
26
CLK_get
data_get
en_get
valid_i
186
184

Fig. 16

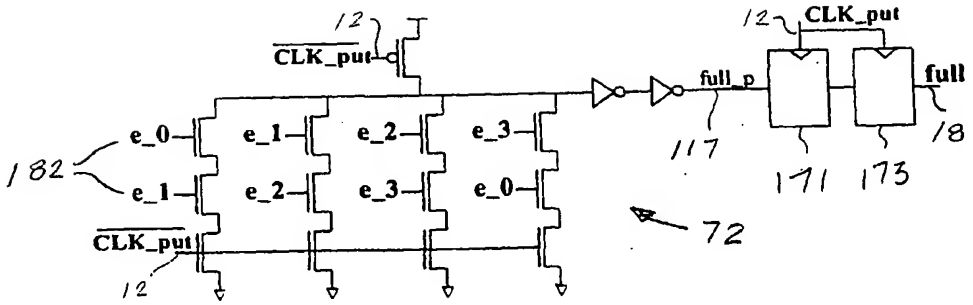

12
CLK_put
12
CLK_put
full_p
full
e_0
e_1
e_2
e_3
182
e_1
e_2
e_3
e_0
117
171
173
18
CLK_put
72
12

FIG. 17

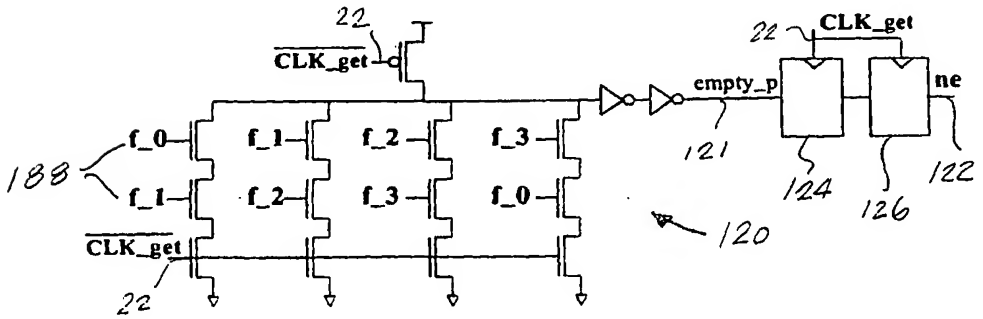

22
CLK_get
22
CLK_get
empty_p
ne
f_0
f_1
f_2
f_3
188
f_1
f_2
f_3
f_0
121
124
126
122
CLK_get
120
22

FIG. 18

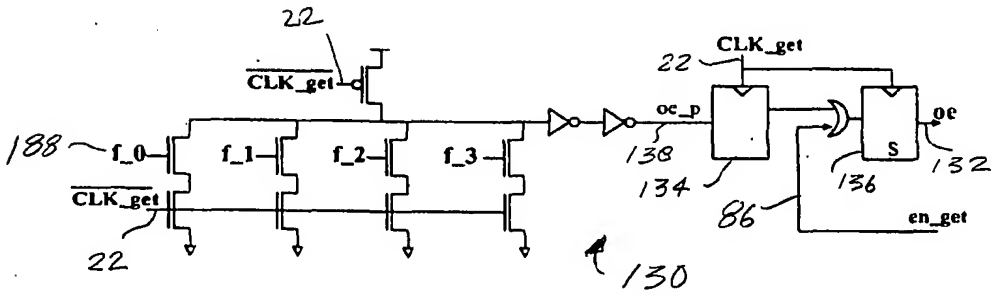


FIG. 19

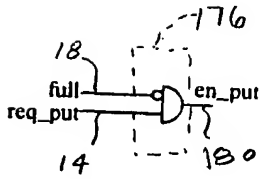


FIG. 20

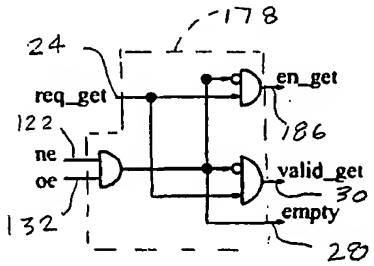


FIG. 21

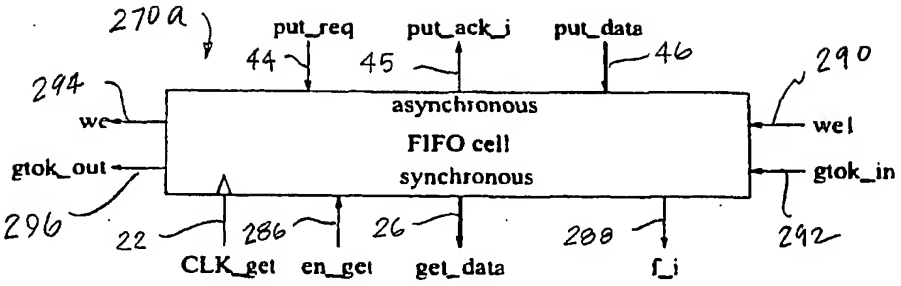

270a
put_req
put_ack_i
put_data
44
45
46
294
290
we
asynchronous
FIFO cell
synchronous
we1
gtok_out
gtok_in
296
22
286
26
288
292
CLK_get
en_get
get_data
f_i

FIG. 22

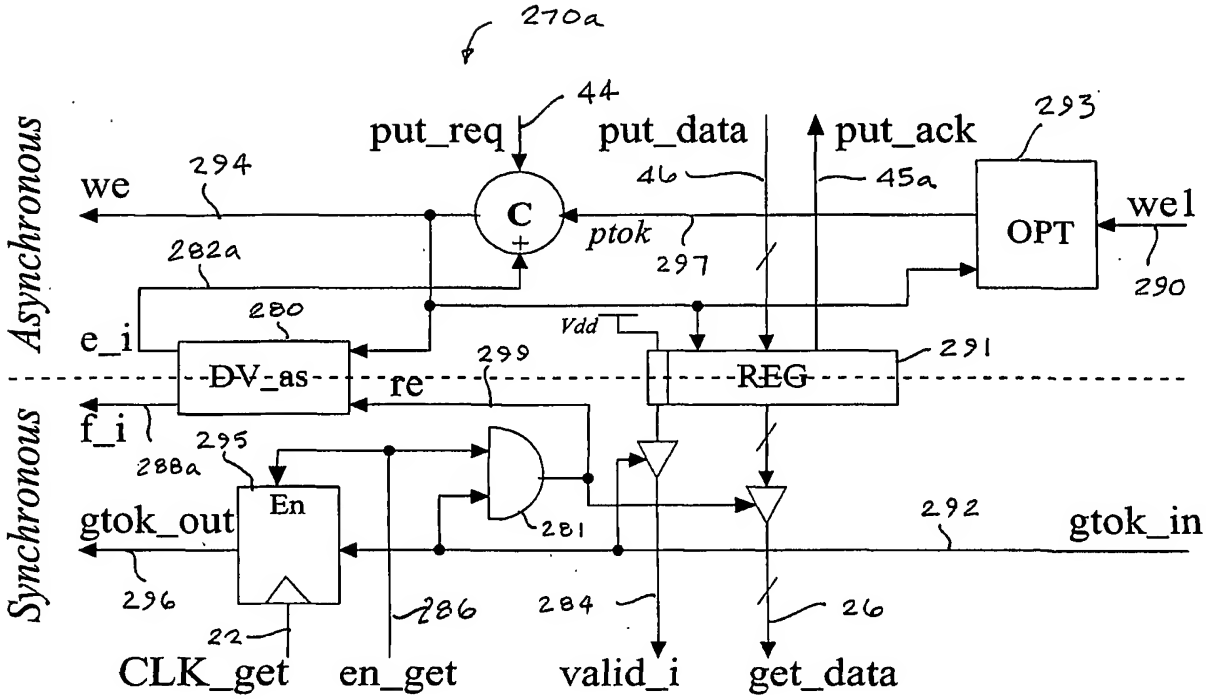

270a
44
put_req
put_data
put_ack
293
294
we
46
45a
C
ptok
297
OPT
wel
290
282a
280
Vdd
e_i
DV_as
299
re
REG
291
Asynchronous
Synchronous
f_i
295
288a
En
281
gtok_out
292
gtok_in
296
286
284
26
22
CLK_get
en_get
valid_i
get_data

Fig. 23

(15/27)

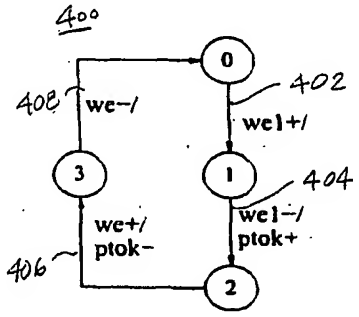


FIG. 24

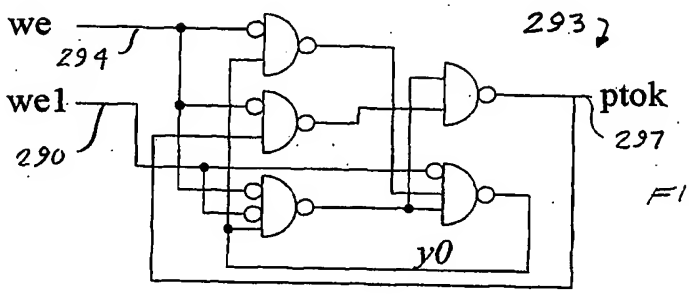


FIG. 25(a)

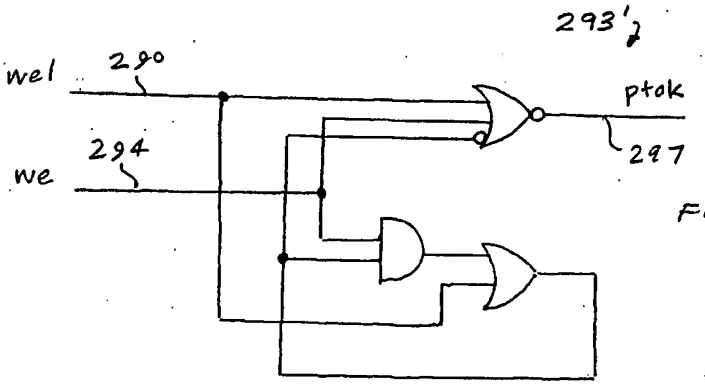


FIG. 25(b)

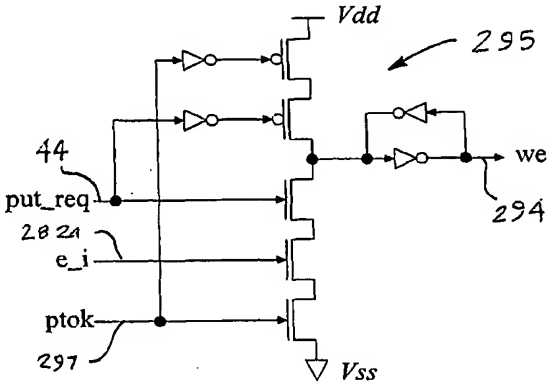


FIG. 26

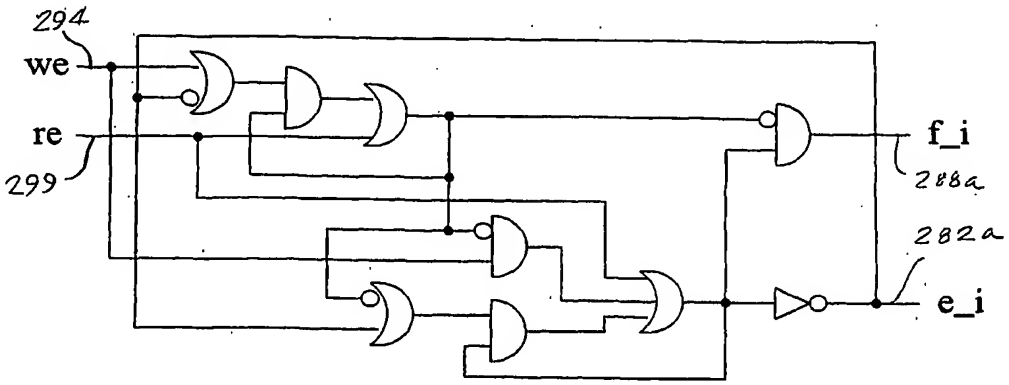


FIG. 27

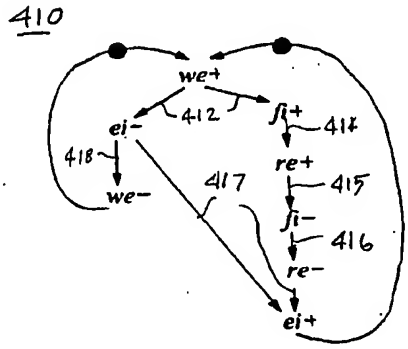


FIG. 28

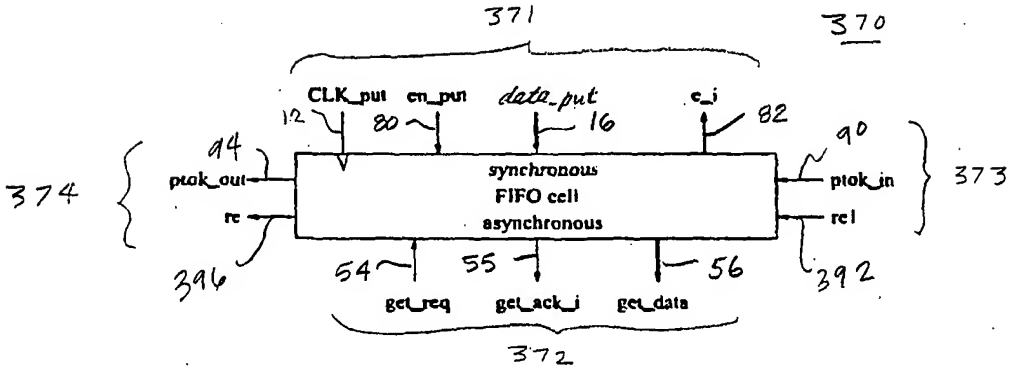

371
370
CLK_put
en_put
data_put
e_i
12
80
16
82
94
90
374
ptok_out
synchronous
FIFO cell
asynchronous
ptok_in
373
re
rel
396
54
55
56
392
get_req
get_ack_i
get_data
372

FIG. 29

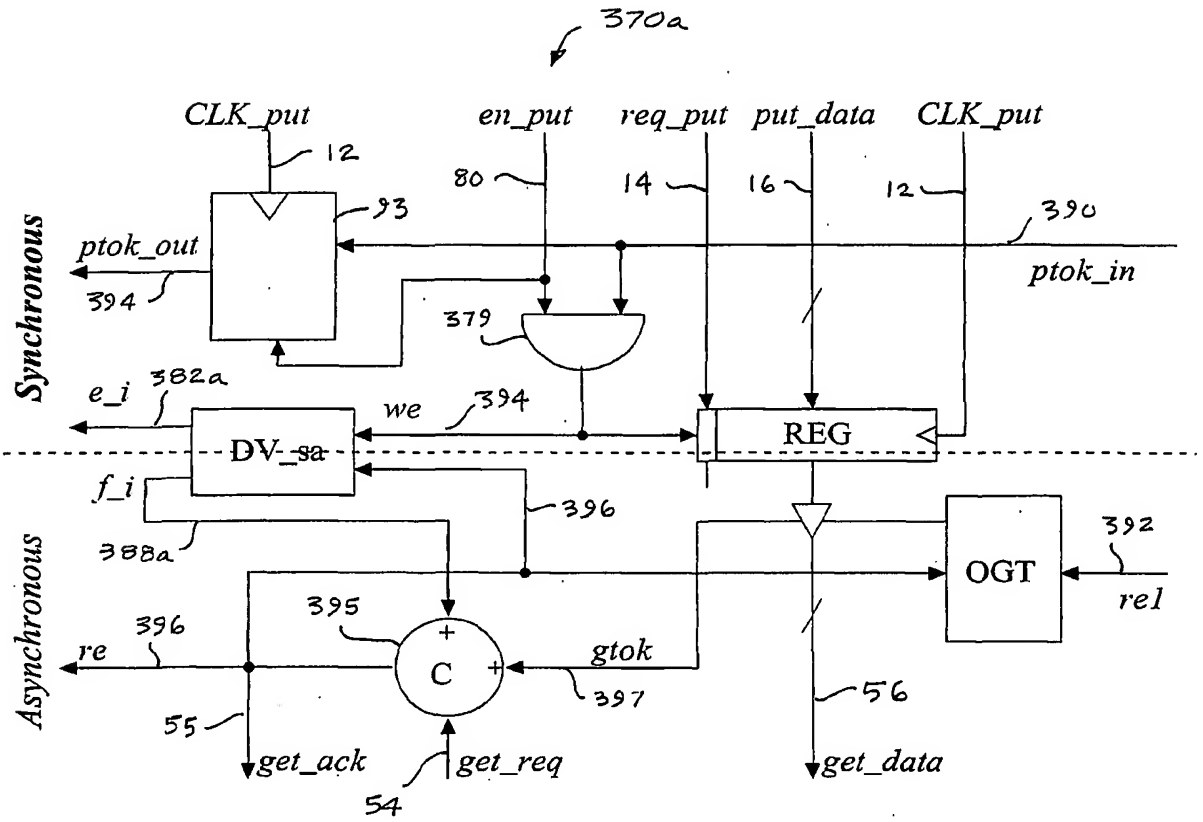

370a
CLK_put
en_put
req_put
put_data
CLK_put
12
80
14
16
12
390
93
ptok_out
394
ptok_in
379
Synchronous
e_i
382a
we
394
DV_sa
REG
f_i
396
388a
392
OGT
rel
Asynchronous
396
395
re
C
gtok
397
56
55
get_ack
get_req
54
get_data

Fig. 30

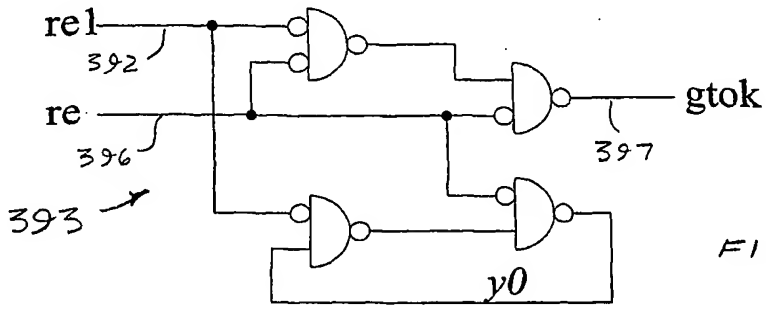


FIG. 31(a)

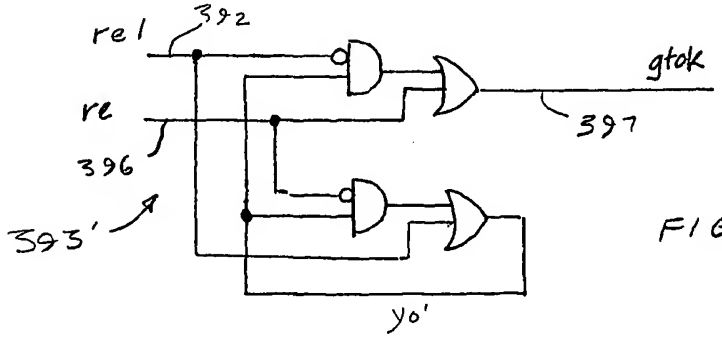


FIG. 31(b)

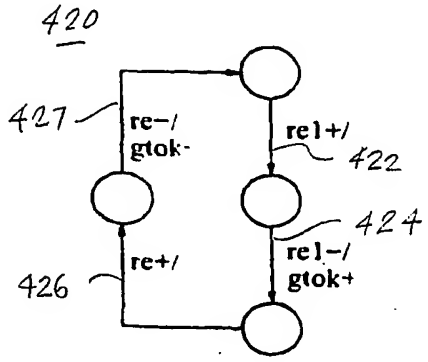


FIG. 32

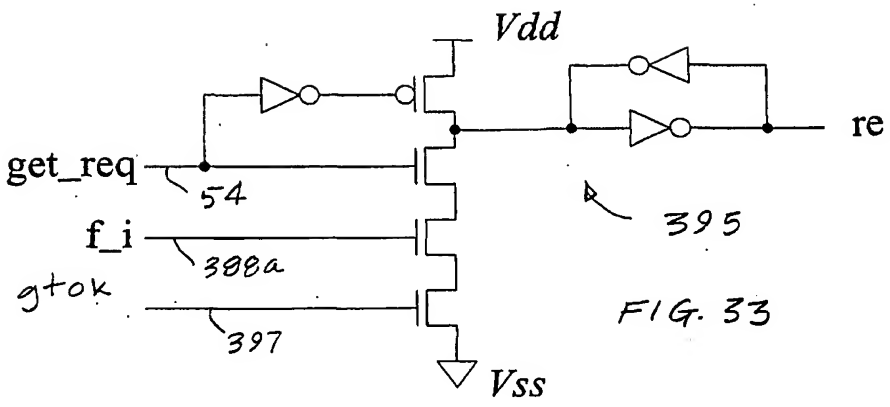


FIG. 33

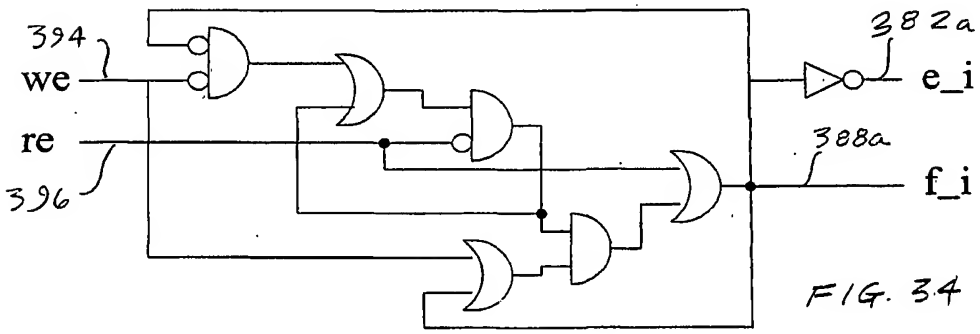


FIG. 34

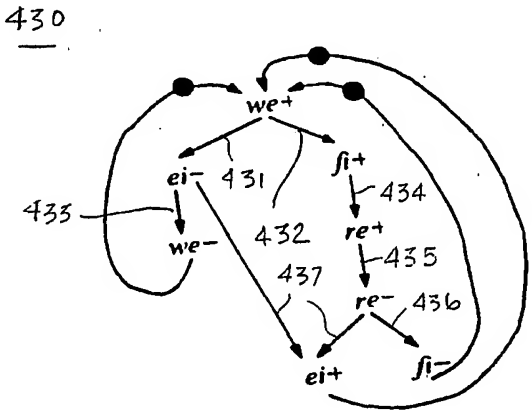


FIG. 35

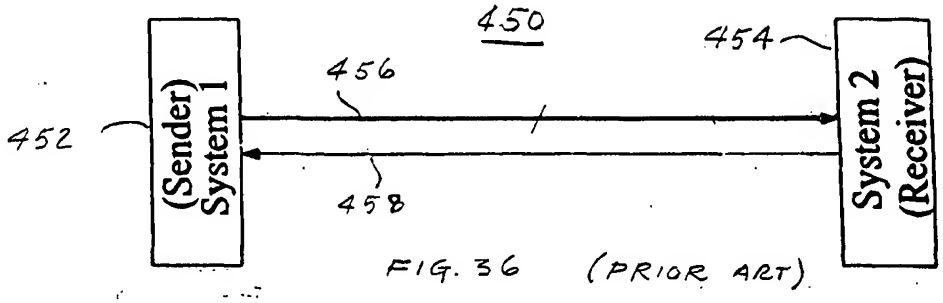


FIG. 36 (PRIOR ART)

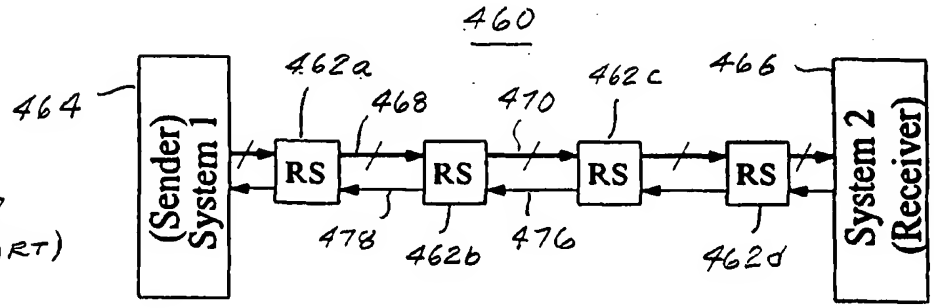


FIG. 37
(PRIOR ART)

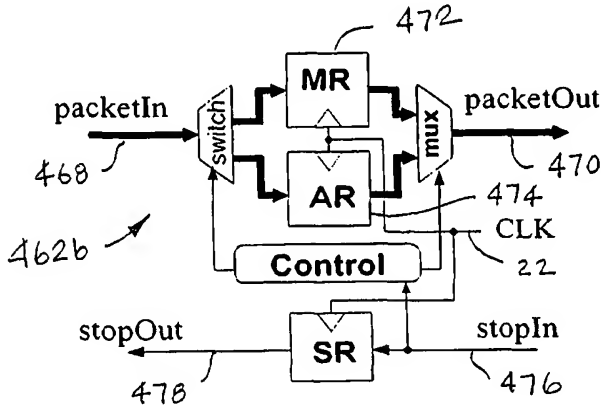


FIG. 38
(PRIOR ART)

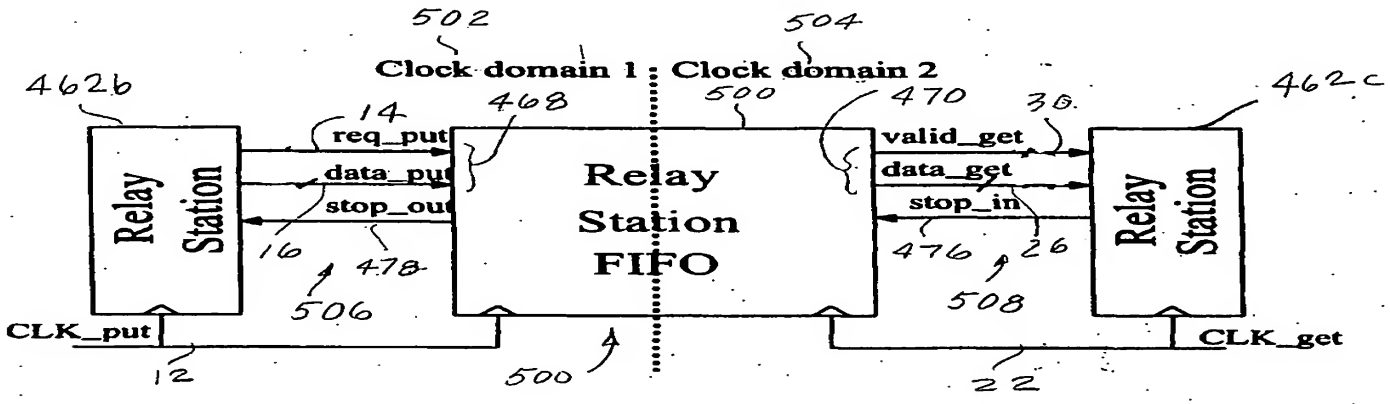


FIG. 39

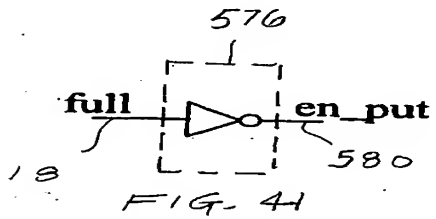


FIG. 41

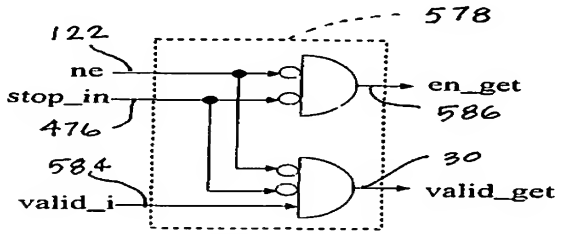


FIG. 42

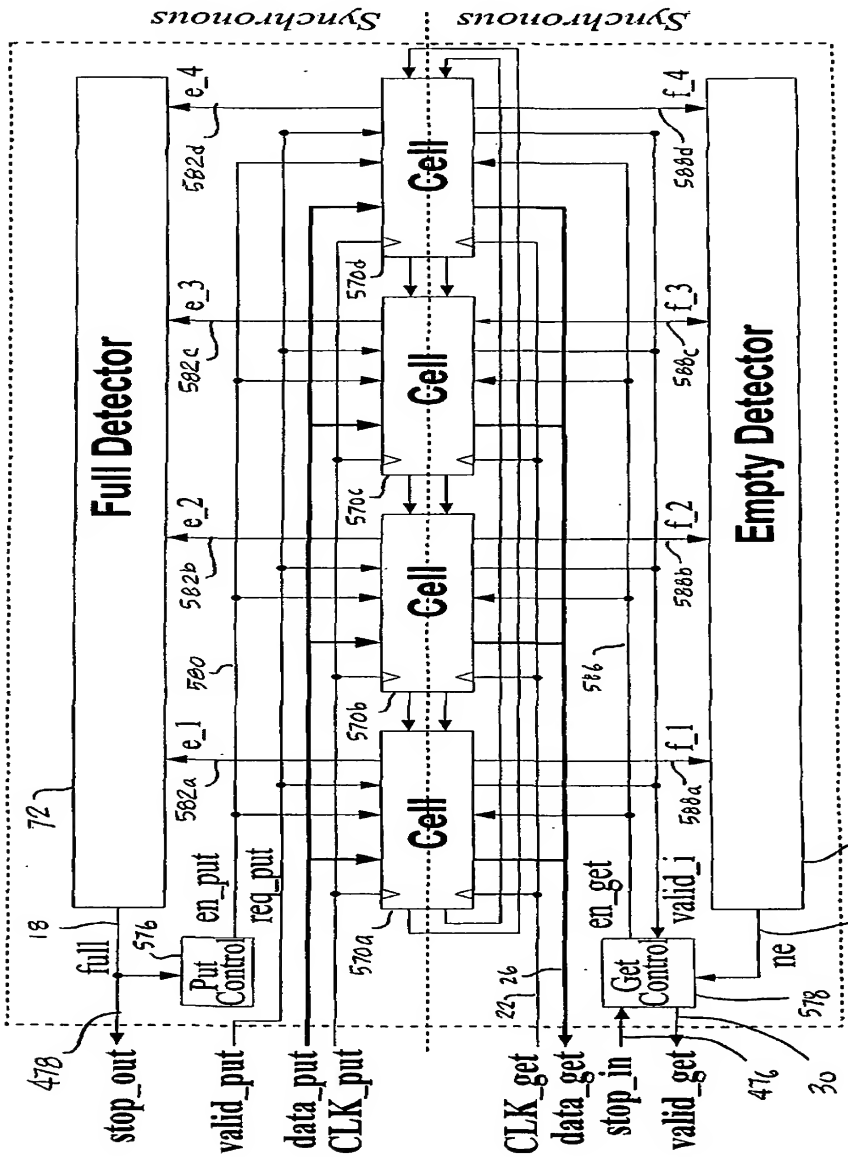

500
Synchronous
Synchronous
Full Detector
Empty Detector
Cell
Cell
Cell
Cell
Put Control
Get Control
full
en_put
req_put
en_get
valid_i
ne
stop_out
valid_put
data_put
CLK_put
CLK_get
data_get
stop_in
valid_get
e_1
e_2
e_3
e_4
f_1
f_2
f_3
f_4
72
18
576
580
582a
582b
582c
582d
570a
570b
570c
570d
588a
588b
588c
588d
574
578
478
476
30
22
24
26
122

Fig. 40

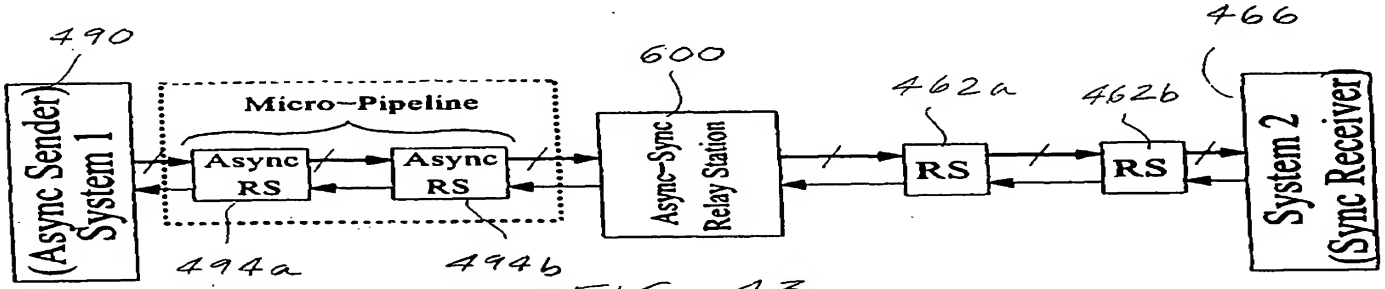


FIG. 43

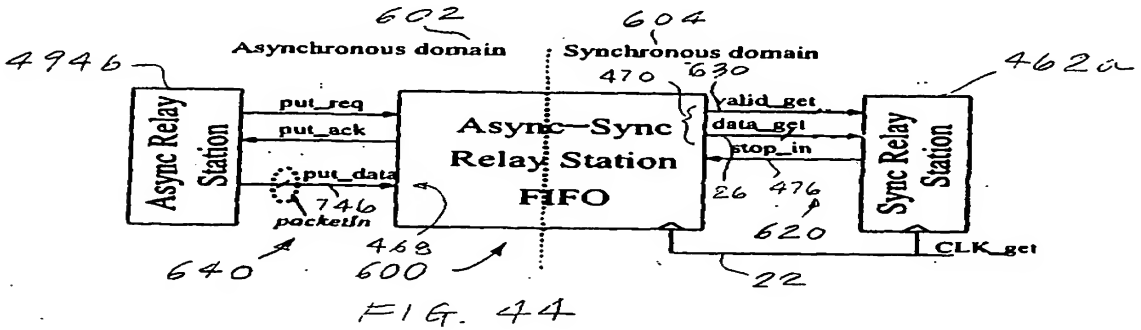


FIG. 44

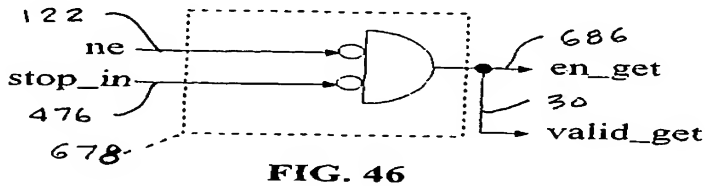


FIG. 46

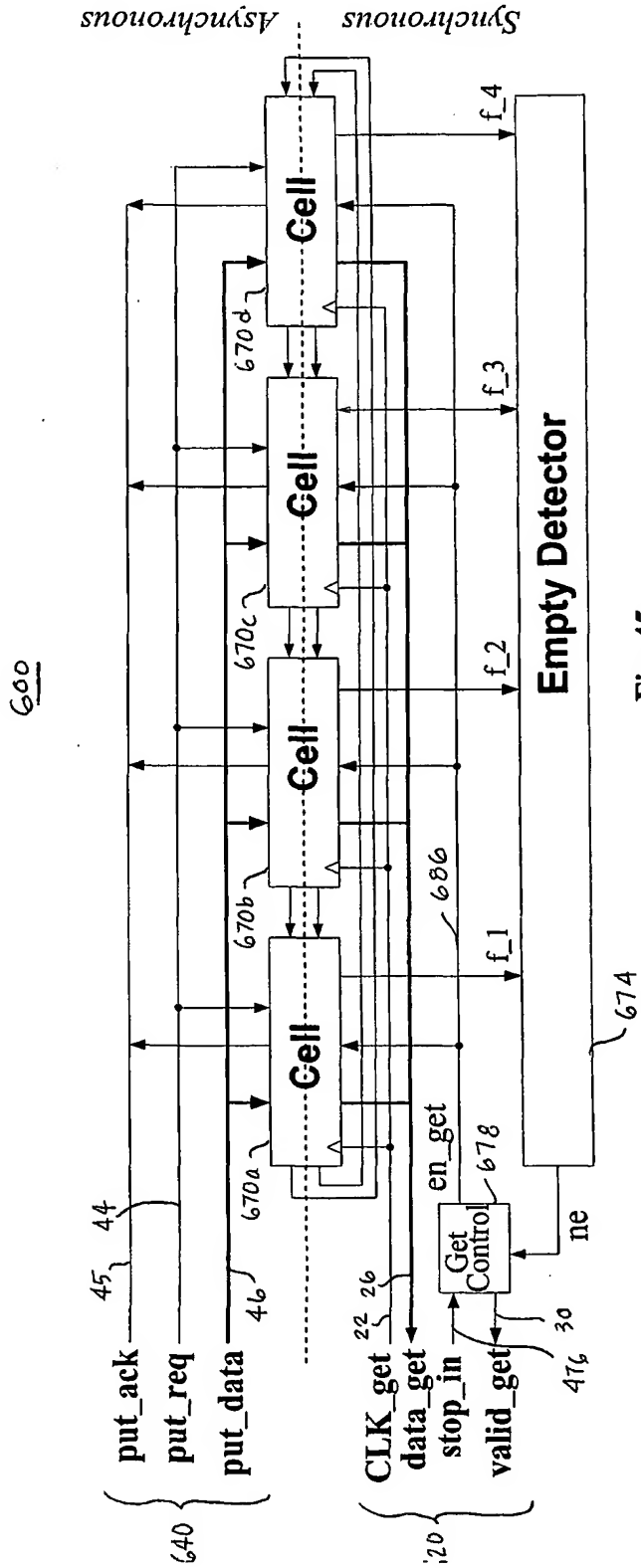

Asynchronous
Synchronous
600
Cell
Cell
Cell
Cell
670a
670b
670c
670d
f_1
f_2
f_3
f_4
686
674
Empty Detector
en_get
678
Get Control
ne
put_ack
put_req
put_data
CLK_get
data_get
stop_in
valid_get
44
45
46
22
26
30
476
640
620

Fig. 45

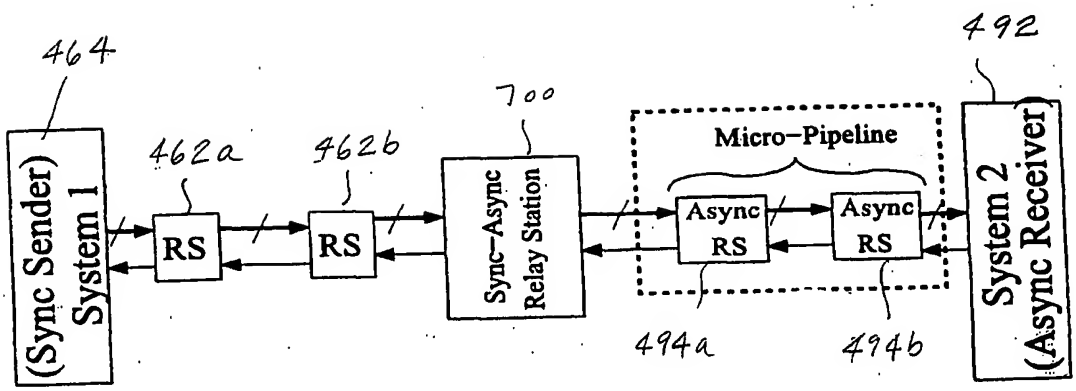


FIG. 47

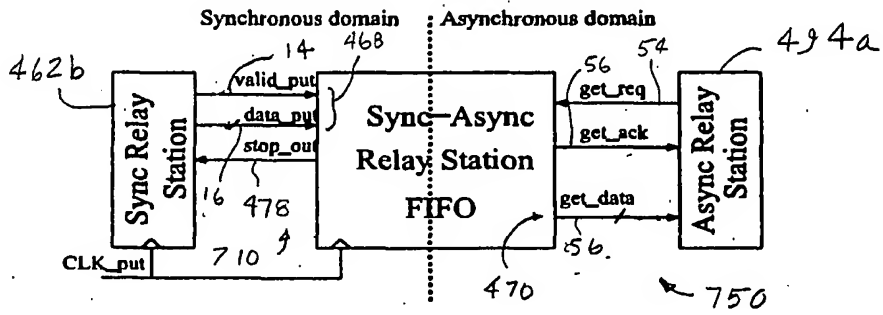


FIG. 48

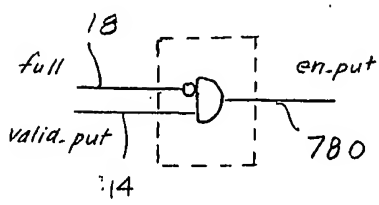


FIG. 50

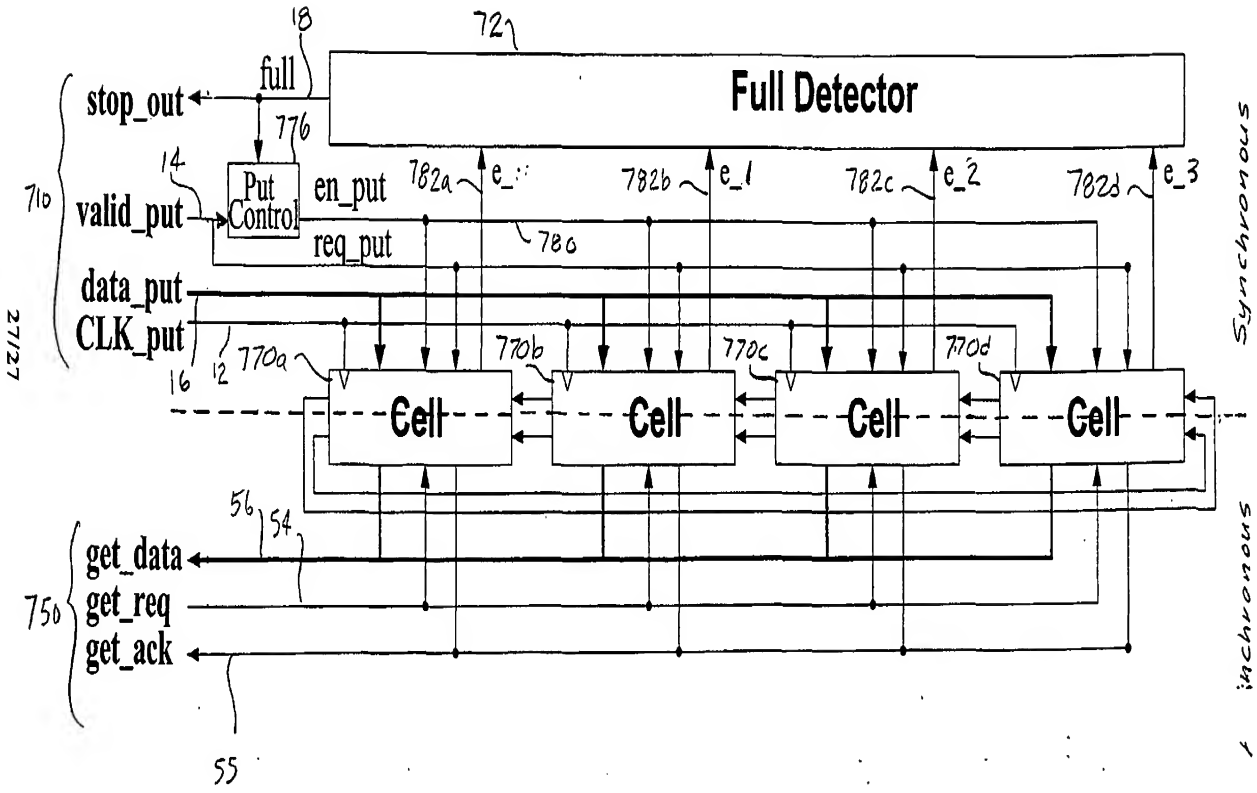

700
18
727
full
Full Detector
stop_out
776
14
Put Control
en_put
782a
e_:
782b
e_1
782c
e_2
782d
e_3
710
valid_put
req_put
786
data_put
CLK_put
16
12
770a
770b
770c
770d
Cell
Cell
Cell
Cell
Synchronous
56
54
get_data
get_req
get_ack
750
55
27/27

FIG. 49